Summary

One of the areas of study within the School of Computing is that of scheduling.
In order to perform scheduling of train driver shifts, the original train data needs to be manipulated into a format that can be used by scheduling software.

The aim of this project is to, create a system to import the original train data, allow the user to manipulate the data manually, include a number of functions for automatically processing the data and finally to have a system of exporting the data for use by the scheduling software.

This report explains how the project was carried out and how the requirements were met, starting from the point of gathering the requirements through to implementation and testing of the final system.

Acknowledgements

I would like to give thanks to Dr Raymond Kwan, my supervisor for the project, for his assistance throughout the project. I would also like to thank Ignacio Laplagne who gave invaluable help in the final testing of the system.

Most of all I would like to thank my wife who encouraged and supported me giving me motivation throughout the project.

Contents

1. Introduction

The introductory chapter gives an overview to the topic of scheduling, which although not central to the project is the driving requirement. The chapter also explains the current use of scheduling software and the data that feeds it within the School of Computing. Finally it explains what improvements can be made and in particular the aims of the project to improve the current situation.

1.1 A Background to Scheduling

There are a number of different areas within scheduling such as Bus Scheduling and Train Scheduling and within each sector there are further considerations. Within train scheduling there is the issue of train time scheduling, distribution of train loads and train driver scheduling. This project looks specifically at the issue of train driver scheduling, defined by Kwan (2004) as "a process that determines the composition of a set of driver shifts for a day's transport operation requiring coverage by drivers". Kwan shows in Figure 1.1 how this fits in with the overall process of train scheduling.

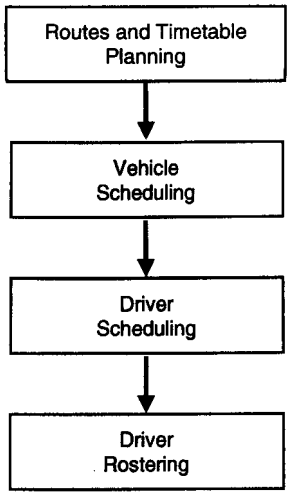


Figure 1.1 - The process of train scheduling

Work on automatic scheduling began in the early 1970's but work on train driver scheduling has only been going since the early 1990s. There can be a large number of variables to take into account when scheduling and a number of algorithms have been designed such as evolutionary algorithms in order to find the most efficient solutions. Details of these algorithms and there application to scheduling are not discussed further here as they are not relevant to the project.

1.2 Current application in the School of Computing

The study of scheduling is carried out within the School of Computing. Working closely with ScotRail and other train companies research is carried out in order to improve the efficiency of train time-tabling software. The notable piece of software produced in the department is the TrainTracs scheduling software, which is used for the scheduling of train drivers and is also used by ScotRail for the same purpose. In order to produce a timetable the TrainTracs system needs a large volume of information, obtained from a number of sources. The data needed by the TrainTracs system is fed in via the two sources as seen below in Figure 1.2.

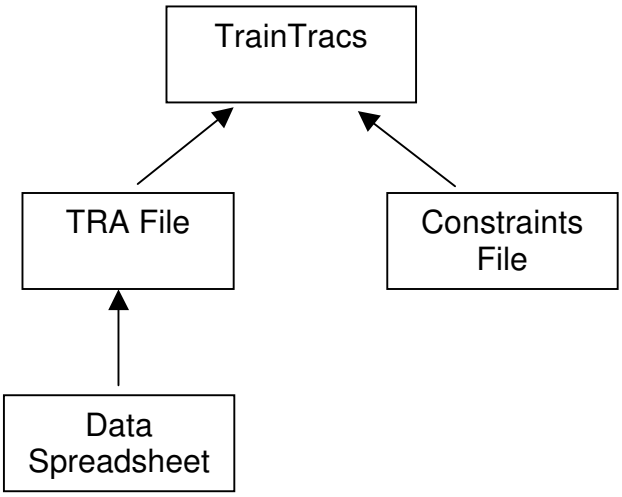


Figure 1.2 - Sources of data for the schedule software

A schedule is generally restricted by a number of constraints or rules, such as the number of hours a driver can work in a day. This information is held in a file and read into the scheduling system shown in Figure 1.2 as the constraint's file. The second file holds the train times. A previous scheduling process carried out by the train company creates this data. The data is then passed onto the department in the form of a Microsoft Excel Spreadsheet to be used for their research. This is referred to in Figure 1.2 as the data spreadsheet.

The train file needs to be manipulated in such a way that the scheduling software can use it. Firstly, the locations at which drivers can be changed on a route have to be specified. Once these locations are chosen then route sections can be

specified. A route section is a segment of any journey from one driver change location to another. Each route section needs a set amount of knowledge required by any driver to travel on it. The required knowledge must be specified for each section.

Once these processes have been carried out the file then needs to be changed into a format that the scheduling software can use and is shown on Figure 1.2 as the TRA file.. The process of converting the original train data into the data file used by the scheduling software is currently carried out with a mixture of manual updates to the spreadsheet and by a set of visual basic macros.

The main problem with this system is that it is not user friendly. When using scheduling software there is often the need for using different scenarios. Kwan (2004) says "it may be necessary to produce many alternative sets of driver schedules for different what-if options." Having to update the excel spreadsheet to create these 'what-if' options can be time consuming. Keeping the data in an excel spreadsheet also opens up the possibility for making the different sheets of data inconsistent with each other. The system could therefore be vastly improved by having a user interface which allows for the updating of data in an easy format and to store the data in such a way that the risk of inconsistencies is reduced or eliminated.

The other issue is one of efficiency. The fact that the data is held in a spreadsheet means that a certain amount of data is repeated. It would therefore be more efficient to hold the data in a normalised database to eliminate duplication and in doing so this would also reduce the risk of inconsistencies within the data as mentioned above.

1.3 Project Proposal

In light of the issues raised with the current process of holding data, the aims of the project seek to solve these issues in the following ways:

1. Review and document the current practices surrounding the storage and provision of data used by the traintracs system.
2. Produce a system that can replace the current system within the School of Computing, with a view to being used within the industry at a future date.
3. Provide an efficient way of holding data.
4. Provide a user interface to allow for data manipulation.

5. Design and implement a set of functions for the importing of data from the excel spreadsheet.
6. Provide a utility for the exporting of data to the TRA file.
7. Provide a user manual for the designed system.

2. Analysis and Methodology

This chapter looks at the requirements of the users in regards to the system and takes into account the processes involved. In light of this information it then discusses the methodology to be employed in the design and implementation of the actual software.

2.1 Background Analysis

It is important when first starting to think about creating a system to understand the background to the project and the data involved. Requirement gathering involves obtaining a full understanding of the current system. Johnson (2002) highlights a set of techniques called O.R.I.Q.S. which may be used to gather the software requirements. The set of techniques are listed as follows:

- Observation.
- Reading/Research.
- Interviews.
- Questionnaires.
- Sample Documents.

It is not always necessary to use all of the techniques in conjunction. There were three techniques that were most applicable to the schedule formatting system. These were Reading/Research, Interviews and Sample Documents.

The first technique was used to understand the terms used within the industry and the processes involved with scheduling. In particular "Handbook of Scheduling" by Kwan (2004) was helpful in giving an understanding of the terms used within the train industry. A summary of these words is given below:

- Diagram - A train's complete journey for one day.
- Point - A location that a train passes through.
- Relief Point - A location where train drivers can change.
- Route Section - A section of a diagram's journey between each relief point.
- Route Knowledge - A code used to denote the knowledge needed by a train driver to be able to go along a certain route section.

The second method was to interview those requesting the system. A number of interviews were carried out in order to gain a specific understanding of the required system. These took place over the course of the year with Professor Raymond Kwan.

Finally, to understand the data being used, sample documents were looked at in order to gain an understanding of the data involved in the process. These being the raw data imported into the system, an example of which is shown in appendix E. A copy of how the exported data should be represented was also studied. From this data and subsequent interviews the following definitions were obtained for data used in the scheduling process:

- Engine Code - The engine code determines what type of engine is used in the train. There are only around six different types of engine. The times for certain train procedures such as turning the engine on and off, are dependent on the type of engine used.
- Operation Code - The operation code denotes what days the diagram runs on. For example Weekends only or Monday to Thursday.
- Subset Code - The subset code is not used within scheduling but is to be held as it is required by the industry.
- Added Time Code - The added time code denotes if there is any extra time taken up at a relief point for events such as starting up the engine.

2.2 User Requirements

In modern analysis the most popular tool for capturing user requirements is the unified modelling language known as UML. Bennett et al (2001) describe UML as "a visual language that provides a way for people who analyse and design object-oriented systems to visualise, construct and document the artefacts of software systems." The language is made up of a number of types of diagram that allow a system to be modelled from its very basic design to completion. The first of these is the 'use case diagram'. Jacobsen et al (1999) state that use cases must be identified for a project to be successful. They benefit a project as they give a clear high level picture as to what is required of a system. This is especially good for large systems as it reduces the paperwork involved in documenting the requirements and is easy to understand when many different types of user are involved in different processes.

As there is only one type of user the need for a graphic display of the use cases is not

required. Instead, the method of use case descriptions as described by Bennett et al (2001) was used to describe the user requirements for the system as shown below.

Use Case - Import data

	Actor	System
1	The user selects the import option	The system checks with the user that they wish to proceed
2	The user either selects no and exits or selects yes to proceed	The system asks the user to specify a file to import
3	The user selects a file to import	The system imports the data giving any details of errors and confirms when complete

Use Case - Select Relief Points

	Actor	System
1	The user highlights points to change and clicks button to make change	The system changes points to relief or non relief points

Use Case - Create Route Sections

	Actor	System
1	The user selects option to create route sections	System creates all the possible route sections based on selected relief points and diagram data. If section already exists then it is not deleted so as to retain route knowledge if entered

Use Case - Add/Delete Coupled Information

	Actor	System
1	The user selects a coupled record to add/delete	The system makes the amendment. The system must ensure that the data is kept consistent with related trains

Use Case - Amend diagram data

	Actor	System
1	The user amends diagram data	The system carries out specified amendment. Will not allow duplicates and will delete related information where appropriate.

Use Case - Export data

	Actor	System
1	The user selects the export option	The system asks the user to specify a file to export to.
2	The user specifies a file to export to	The system prompts user to add a title
3	User adds title or leaves blank	The system confirms when complete and asks user if they wish to view exported file
4	User selects to open file	The system opens the file for the user

The method above produces details specific to the user however in order to produce the software it is necessary to define requirements specific to the system. A method for accomplishng this has been defined by European Commision (2001) defined below.

Data Stored

- The database will store a set of diagrams with the fields: Diagram Code, Engine Code, Operation Code and Subset Code.
- The database will store a set of journey times for each diagram with the fields: point, arrival time, departure time, train type code, if a driver is needed, added time code.
- The database will store all the possible route sections based on the diagram data with the fields: from point, to point, route knowledge.
- The database will store a set of via points for each route section with field: point.
- The database will store a set of times taken for set processes for each type of engine with the fields: train type, train type code, immobilisation time, mobilisation time, preparation time, disposal time.

System Requirements

- The database will store which trains are coupled together.
- The system will allow relief points to be selected.
- The system will allow data to be imported into the database.
- The system will highlight any errors occurred whilst importing.
- The system will allow the user to specify the location of the file to be imported.
- The system will allow route sections to be created from the imported data.
- The system will allow the user to view route sections with no route knowledge applied.
- The system will allow the user to export data in specified format.
- The user will be able to select the file to export to.
- The system will highlight the stages of progression towards exporting the data.

Constraints

- The system will not allow the user to amend the route sections.
- The system will not allow trains to be coupled unless journey ties are identical.
- The system will not allow diagrams with the same diagram code.
- The system will not allow data to be exported unless relief points have been selected, route sections have been created and route knowledge has been added.

Non Functional Requirements

- The system will allow for the quick update of route knowledge.
- The system will offer a simple interface for updating relief points.
- The system will be easy to use.

The user requirements specified, show what is required of a system but they do not specify the order in which processes are carried out within a system. Bennet et al (2001) describe activity diagrams as "a means of describing workflows". Figure 2.1 shows the main flow of work within the schedule formatting system. This can then be used to plan how the system should be implemented.

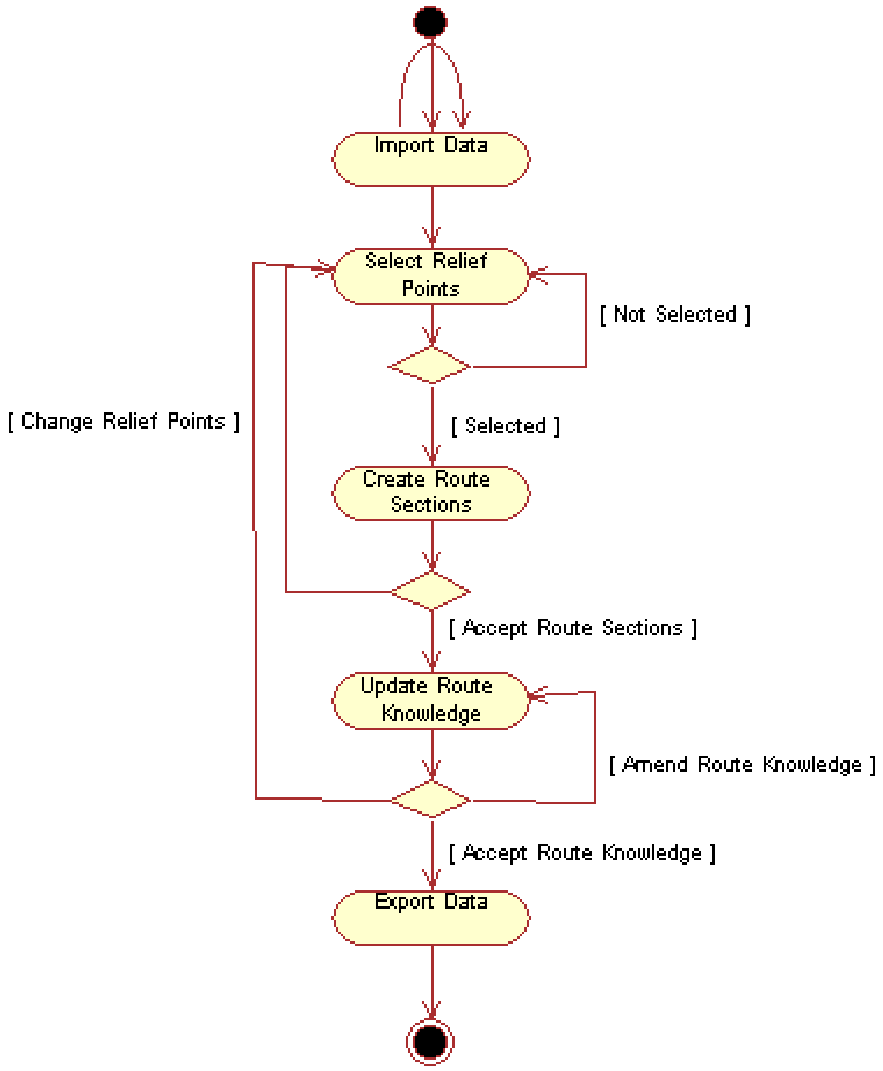


Figure 2.1 An activity diagram showing the flow of work in the system

2.3 Technology

The right choice of technology to employ when creating a system is vital to creating a successful project. Behforooz and Hudson (1996) state "selection of programming languages used to code the software system can have a long lasting effect on software system performance and maintenance". Technologies for the project can be split into two sections, those required for the database and those for the interface. Depending on which of these are chosen for one area can affect the choice of technology in the other area. In order to assess the different possibilities, a number of technologies were investigated, showing their main advantages and disadvantages.

Microsoft Access

Advantages

- A complete package comprising of a database and interface allowing for the whole system to be built in one package.
- The software is already installed within the School of Computing and is common in most work places. This allows the software to be easily deployed.
- Designer already has a good working knowledge of the software.

Disadvantages

- Not designed for large amounts of data.
- Lack of flexibility in creating the user interface.

Microsoft SQL Server

Advantages

- Designed to work with large amounts of data.
- Allows for quick retrieval of data.
- Software already installed within the School of Computing.
- Designer already has good working knowledge of software.

Disadvantages

- Unlikely to be available to users if used outside of School of Computing.
- Needs separate interface component.
- Would need to be conFigured on installation.

MySQL

Advantages

- Designed to hold large amounts of data.
- Open source software. No cost issue for installation outside School of Computing.
- Already installed within School of Computing.

Disadvantages

- Needs separate interface component.
- Would need to be conFigured on installation.
- Lack of usability in creating table relationships.

Visual Basic 6

Advantages

- Can be built into Microsoft Access application.
- Developer has some knowledge of the product.

Disadvantages

- Has become outdated as an interface tool due to introduction of visual studio.net.

Visual Basic.net

Advantages

- Latest Microsoft development tool. Should be around for a long time and will not become outdated.
- Works well with both Microsoft Access and SQL Server.
- Works well with the prototyping method as it allows for the quick building of mock up interfaces.

Disadvantages

- New technology resulting in minimal amount of reference material.
- Minimal knowledge of product by developer.

Initially, the clear choice in terms of a database was to use Microsoft Access. This was due to the fact that the database would not need to be conFigured upon installation and would not need to store large amounts of data, meaning that a powerful database system such as SQL Server was not required. The fact that the interface can also be built in the same package and the developer already had a good knowledge of the system were added positive factors in making the final choice.

The system was first designed in Access but using the evolutionary approach (Discussed in the next section) it became clear that this would not offer the level of functionality and usability required by the users. An alternative was therefore needed. The next best option was to retain Access as the mode for storing the data but to use Visual Basic.net for the interface. This choice was taken because Access was still the best tool to use to store the data and as a result visual basic.net was the best tool to interact with the database. This was due to the fact that they are both Microsoft products and are designed to interact well, and also for the reasons stated above.

2.4 Methodology

Pressman (2000) defines methodology as the process of providing the "technical "how to's" for building software." It provides a method for dealing with each section of the software from requirements analysis to testing. The oldest and most common of these methods is called the Waterfall method, also known as the System Lifecycle (Lauden and Lauden, 2002). This method breaks the process into distinct sections of work carried out in a set order. At the end of these sections of work the initial product is produced. Redmill (1997) defines the sections of work within the method as follows:

- Specification.
- Design.
- Construction.
- Validation.
- Installation and acceptance testing.

Redmill states that the great advantage of the waterfall model is "that it represents basic engineering practice: that specification should precede design, design should precede construction, etc." Laudon and Lauden (2002) however, state that although "The systems lifecycle is still useful for building large complex systems that require a rigorous and formal requirements analysis, the approach is not suitable for many small desktop systems, which tend to be less structured and more individualised". This would suggest then that this method is not suitable for the software in question.

A method more appropriate is possibly one of the evolutionary methods such as the incremental method defined below by Pressman (2000) in Figure 2.2. Pressman says "the incremental model focuses on the delivery of an operational product with each increment." This is useful for the system in question as it can easily be broken down into the following operational products:

- System Database.
- Selection of Relief Points.
- Route Section Knowledge Update.
- Train Data Manipulation.
- Create Route Sections Function.

- Import Function.
- Export Function.

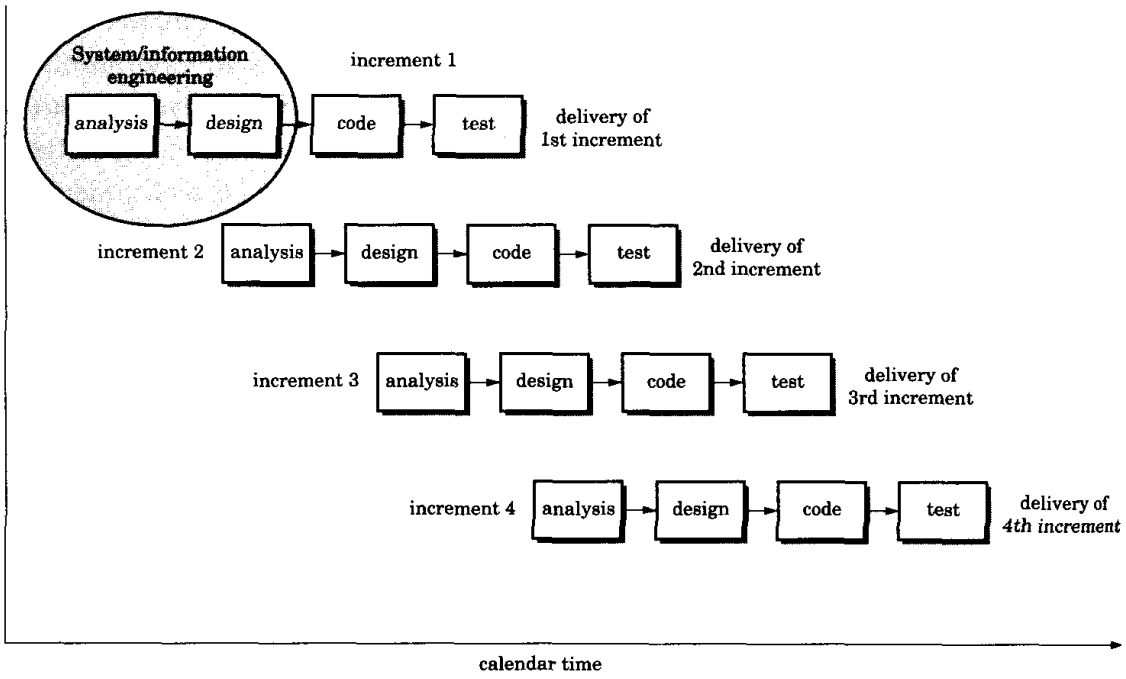


Figure 2.2 - A diagram of the incremental model

Pressman states that the prototype method can be used in conjunction with the incremental model. Prototyping enables the designer to obtain feedback from the user as to the suitability of the produced software

It was therefore decided to use the incremental method, breaking the software into the above products and using the prototype method to gain user feedback. This is reflected in the schedule for the project found in appendix C. This schedule was later updated to incorporate two missing items of work omitted. These were the export function and the testing of the system. These were duly added as shown in appendix D.

3. Design

This chapter looks at the design issues in creating the database and interface for the system. It gives a detailed design specification of the database and looks at the considerations needed in the issue of human computer interaction when designing the interface.

3.1 Database design

Elmasri and Navathe (2000) outline a set of procedures for the designing of databases as follows:

- Requirements analysis.
- Conceptual design.
- Implementation.

Requirements analysis was covered in the previous chapter and we will now look at the conceptual design. Elmasri and Navathe break this down into two steps: defining the entity types and defining relationships between these entities. The defining of the entities for the schedule formatting software is shown below:

Conceptual design

Entity Type	Attributes	Key
Points	Point, Relief, Depot	Point.
Diagram	Engine type, Diagram code, Operation code, Subset code	Diagram Code.
Train Times	From Point, to point, Departure, Arrival	All attributes would make key. Simplified by adding ID.
Route Section	From point, To point, Route Knowledge	No key so ID added.

Relationships

1. Name: Diagram goes Via
   Type: One to many between diagram and train times and many to many between route sections and train times.
   Description: The relationship is between diagram, train times and route section. All participations are total, that is to say that a diagram must consist of a number of train times and that these train times make up a route section and so therefore must contain a number of route sections also. It has the attributes, driver needed, train type code, added time code and order.

2. Name: Goes Via
   Type: One to many
   Description: The relationship is between route section and points. Both participations are total, that is to say that a route section must contain a number of points that must exist in the points table. It has the attributes, point and order.

The participation constraint in a relationship defines whether the existence of an entity is defined by its relation to another entity.(Elmasri and Navathe, 2000). If a participation is total then every entity in one table must be related to an entity in the related table.

As a result of defining these entities and their relationships an ER diagram can now be defined shown in Figure 3.1. The entities are shown as rectangles with the relationships represented in the diamonds. The ER Diagram gives a simple and clear view of the ER schema.

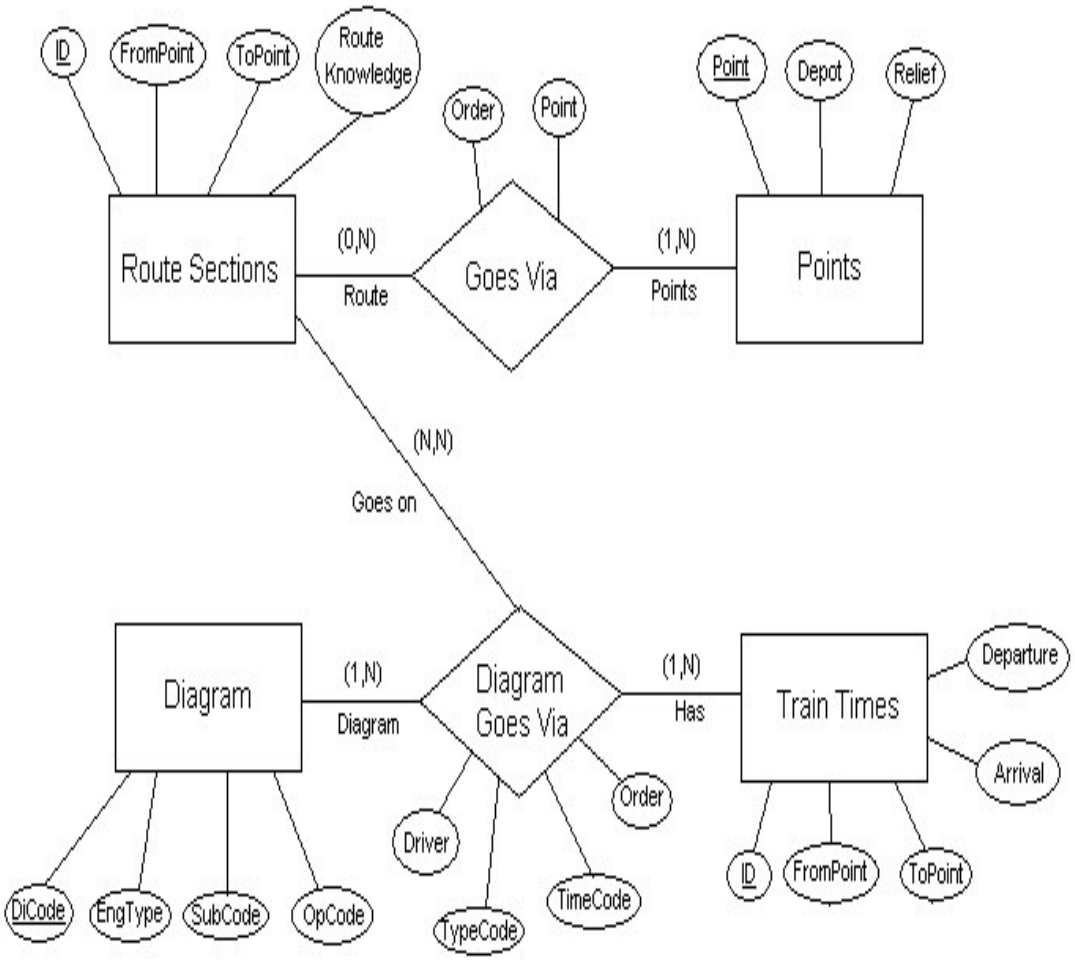


Figure 3.1 - ER Diagram of database used in Schedule Formatting software

In summary the tables for the database can be defined as follows.

Table	**Attributes**
Diagram	DiCode, EngType, SubCode, OpCode
Diagram Via	DiCode, TrainTimeID, RouteID, Driver, TypeCode, TimeCode, Order
Train Times	ID, FromPoint, ToPoint, Arrival, Departure
Route Sections	ID, FromPoint, ToPoint, Route Knowledge
Route Section Via	RouteSectionID, Order, Point
Points	Point, Depot, Relief

Figure 3.2 - Database Tables

We now have a full relationship schema and table definitions, and can therefore go on to carry out normalisation.

3.2 Normalisation

The normalisation process applies a set of rules to the relation schema in order to check whether the schema is in a normal form. By satisfying this normal form we can be assured that we have minimised redundancy and the possibility of anomalies caused by insertion, deletion and update of the database (Elmasri and Navathe, 2000)

Elmasri and Navathe (2000) define four normal forms to be used when assessing a relational schema as shown below:

- **First Normal Form**
  First normal form specifies that all attributes within a database must only include atomic values. This means for example that a name should be split into forename and surname.
- **Second Normal Form**
  Second normal form looks at the subject of full functional dependency. All non-key attributes must be dependent on the whole key.
- **Third Normal form**
  Third normal form deals with the issue of transitivity. If an attribute (x) is dependant on an attribute (y) and if an attribute (z) is dependant on attribute (x) then transitivity has occurred and the table should be broken into two further tables.
- **Boyce-Codd Normal Form**
  This normal form is similar to third normal form. The difference is that if an attribute (x) is dependent on an attribute (y) then (y) must be a superkey. A superkey is an attribute or attributes making up the key of a table.

From this definition it is clear that before the schema can be tested the functional dependencies need to be defined. Elmasri and Navathe (2000) describe functional dependencies as "a constraint between two sets of attributes from the database".
The functional dependencies for the schedule formatting software are given below:

- DiagramCode → { EngType, SubCode, OpCode}
- {DiCode, TrainTimeID} → { Driver, TypeCode, TimeCode, Order}

- ID → {FromPoint, ToPoint, Arrival, Departure}
- ID → {FromPoint, ToPoint, Route Knowledge}
- {RouteSectionID, Order} → Point
- Point → {Depot, Relief}

By applying the rules referred to in Figure 3.2 we can see that all forms are met and the database is fully normalised.

3.3 Denormalisation

Reese (2003) states that "fully normalised databases can require complex programming and generally require more joins than their unnormalized or denormalized counterparts." During the implementation phase for each of the operational products defined in Chapter 2 it was found that having the database fully normalised did cause complications and thus it needed to be denormalised. This process can be defined as "reducing the join of a higher normal form to a lower one." Elmasri and Navathe (2000). This allows for the system to be at a high level of normalisation as a whole, with certain areas at a lesser extent. The designer can then implement rules and constraints to combat any inconsistencies that might be caused due to the denormalisation of the schema. The process in particular to the schema for the schedule formatting system will now be discussed.

The train times was created as an entity rather than a relationship due to the fact that a number of trains can be coupled together between any two points. A diagram can therefore share a train time with another diagram, which resulted in the train times needing to be an entity. In doing so a train time would have to be created separate to a diagram and then assigned to the diagram. This would be impractical for the user when they are updating the database. A different design therefore needs to be applied. A more user-friendly approach would be to have the train times as a relationship between the diagram and points tables. A journey could then be added directly against a diagram. Making this change however, does have negative results. Unlike with the previous design there is now no ability to represent which trains are coupled together. In order to overcome this problem a new relationship was added to the database schema entitled coupled.

The coupled table specifies which trains are coupled to a train on a journey. A coupled journey will therefore be stored at least twice, one record for each train involved in the

coupling. This means that if a coupled train record were removed for one train then an anomaly would occur in that the related train record would still have a coupled record entered against it. In order to avoid these anomalies further constraints are required at the interface level. This will be covered in the Implementation Chapter.

The second change to the schema is due to the route section creation process and the exporting of data for the schedule software. When the data is exported only the journey times for the route sections are required along with the route knowledge for that section. With the database in its current format there is no direct link between the diagram and the route sections it is composed of. Figure 2.1 in Chapter 2 shows that the route sections are created from the diagram information and relief point selection process. With the current design, when the export takes place the diagram information would have to be analysed similar to that in the route selection process in order to calculate the route sections that make up a diagram's journey. A simpler approach would be to store the route sections for each diagram in a separate table that can be referred to when exporting. This table could easily be created when the route sections are created. In doing so it would reduce the time taken to export the data to file.

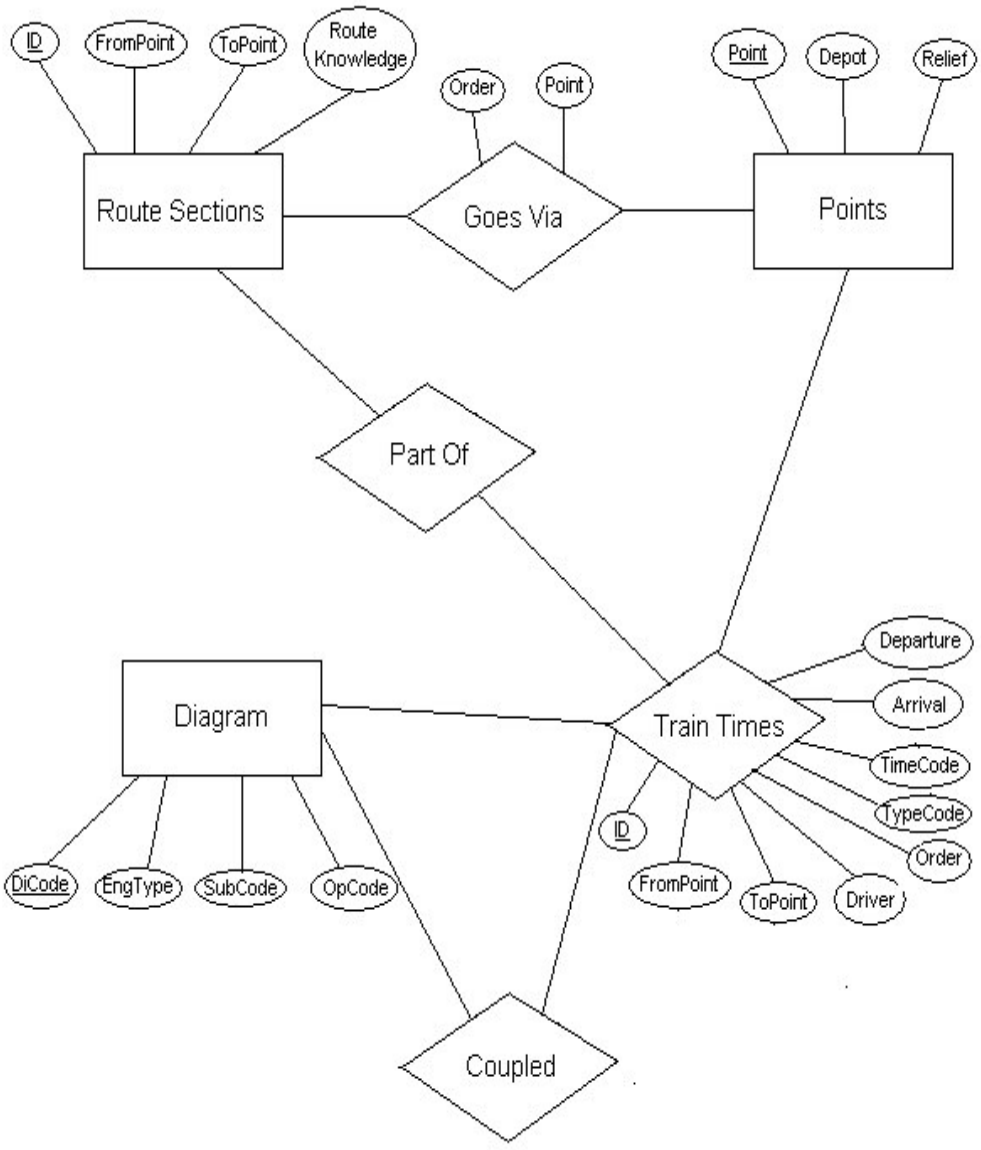


Figure 3.3 - New ER Diagram

Figure 3.3 shows an updated ER diagram taking into account the changes made in the denormalisation process.

3.4 User Interface

When designing a system Shackel (1997) states that it "must start with the end-users and be user-centered around them. Therefore, the human factors aspects become paramount." It is therefore true to say that the main issue when designing the user interface is that of human computer interaction.

Carroll (2001) defines the subject of human computer interaction as “understanding and creating software and other technology that people will want to use, will be able to use, and will find effective when used”. Carroll also states that human computer interaction can be best implemented within the prototyping methodology. This is because the fundamental aspect of the prototype model is user feedback and it is therefore fair to say that a certain amount of the design of the interface should be due to user feedback. As discussed in Chapter 2, the process of prototyping has been included in the methodology of the project and the human computer interaction should therefore be reflected in the interface as a result of this process.

An example of this process carried out in the project can be seen in Figure 3.4.

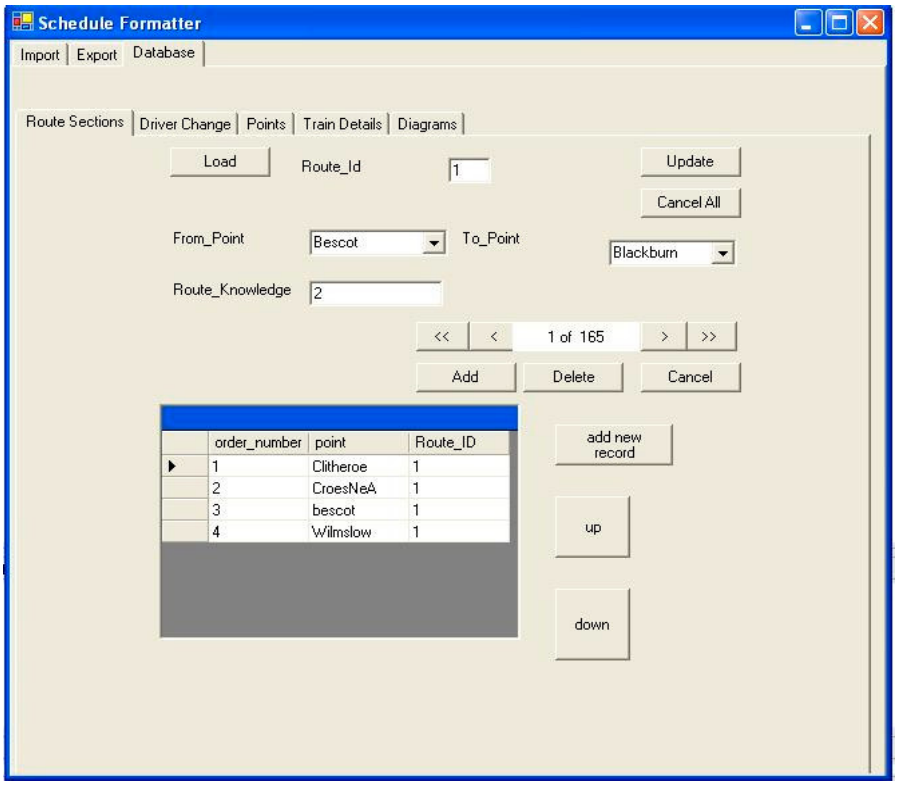


Figure 3.4 - Initial Route Section screen

Figure 3.4 shows an early prototype of the route sections screen. The screen represents one route section at any one time. The user highlighted the fact that adding the route knowledge in this way would be slow and cumbersome as only one record can be viewed at a time. In light of this the screen was amended to show a table of route sections which could be updated in a more efficient manner with all the route sections displayed together on the one screen. This design specification of displaying

all data on the one screen was then applied to all screens from this point on as will be shown in the following chapter.

As well as making the project user centred in order to ensure good human computer interaction there are a number of standard rules that can be applied to an interface to ensure good practice. Neilsen (1993) specifies the following criteria, generic enough to be able to be applied to almost any interface. With each rule an application to the schedule formatter software is given stating how it will be represented in the system.

- **Simple and natural dialogue**
  This relates to issues such as the number of options given to the user at any one time and the amount of text displayed to the user and the way the information is set out on the screen.
  The software will use the standard Microsoft grey background with black text. This is a proven system and is used across most Microsoft software.
  Information will be grouped together in the form of tables. The system is not textual in nature and therefore the rules regarding layout of text do not apply.
- **Speak the users language**
  In most cases this would mean using everyday language. However as we have discussed in the first chapter the rail industry uses specialised terms. These terms have been used throughout the system. Clear language has also been used when providing error messages.
- **Minimise user memory load**
  This process is in regards to displaying what is required of the user clearly at any point within a system. This ensures that the user does not need to remember a large set of rules, procedures and operations. It can be achieved by clearly setting out the options on the screen.
  The system will use a number of tabs to display the screen that the user is currently in and which screens can be selected. The system will also allow the user to view at which stage of the process they are at as defined by Figure 2.1 in Chapter 2.
- **Consistency**
  An applications layout and design should be kept consistent throughout. This ensures that the user can quickly learn and understand how to use an application.

The Visual Studio.net framework allows for consistency throughout an application by using standard Microsoft forms. These have been used throughout the system. The use of standard data tables has also been used throughout.

- **Feedback**

  An application should give feedback to the user to ensure that they understand what the application is doing and why an error might have occurred.

  The system will use message boxes throughout the system to inform the user if they have tried to carry out an illegal task or if an error has occurred. This is especially important in the importing of data where a number of errors could take place due to errors in the data. Further information as to when feedback is given in particular instances is given in the following chapter.

- **Clearly marked exits**

  An application should have clearly marked exists so that the user understands how to exit the application at all times.

  The system will use the standard Microsoft cross in the top right corner. It will also include an exit option in a drop down menu.

- **Shortcuts**

  An application should have shortcuts built in to speed up the selection of processes by the user. These normally take the form of short cut keys.

  The system will have a number of functions that can be selected from a drop down menu. Short cut keys will be provided for each of these functions

- **Good error messages**

  An application should give clear information when an error occurs within the system. This ensures that a user can understand if they have carried out an illegal process or if there is an issue with the system outside of their control.

  The system will have the catching of errors built into the code which in turn will provide feedback to the user of errors occurred in a easy to understand format.

- **Prevent errors**

  An application should have rules programmed into it that do not allow a user to carry out illegal tasks or if a task could result in an error then confirmation can be requested from the user.

  The system will have rules built into it to stop illegal actions occurring, such as disallowing the coupling of two trains with non-matching data. The system will

also ask for confirmation from the user before carrying out critical procedures such as when importing or exporting data.

- **Help**

  An application should provide help to the user either electronically or through paper.

  The system will have a user manual to accompany the system.

4. System Implementation

This chapter discusses the implementation of the project. It first looks at the implementation of the database and then takes each of the operational products as outlined in Chapter 2. It finally looks at the creation of the manual which goes along side the software and the installation requirements.

4.1 Database Implementation

The database was implemented in Microsoft Access as discussed in Chapter 2. The tables were implemented as defined in Chapter 3. Microsoft Access also allows for the definition of relationships between tables and these were implemented as also defined in Chapter 3. Further, Microsoft Access includes the use of cascade updates and deletions as shown in Figure 4.1 to enforce referential integrity. The cascade effect means that when a record is deleted or updated in a table then all records relating to that record in another table will also be updated or deleted. This ensures that anomalies and redundant data are kept to a minimum.

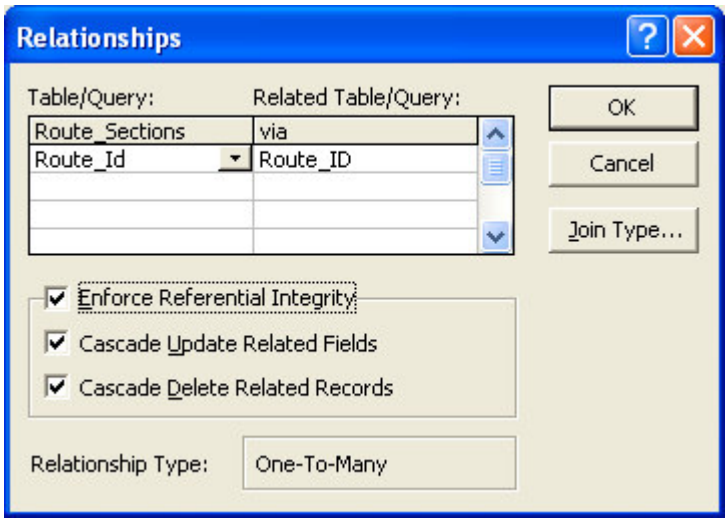


Figure 4.1 - Application of referential integrity through Microsoft Access

Visual Studio.net allows for the creation of datasets when dealing with database records. Tables and relationships can be specified in a similar way to the defining of relationships in Microsoft Access. One of the issues with this is that of cascading deletions, which both Access and Visual Studio.net include. This can create the possibility of concurrency errors if at both the interface and database level records are

automatically being deleted. It was decided therefore to include the cascade facility purely at the database level as this is the permanent storage area.

Bilgin (2002) highlights the fact that large datasets can slow a system down as the whole dataset is updated when saving to the database. The implementation therefore uses a number of small datasets.

4.2 Operational Products Implementation

Each of the operational products was implemented in turn and are now as follows.

4.2.1 Relief Point Selection

The first section of the implementation was to create a system for selecting relief points. As the user may need to change which points are relief points relatively frequently, it was highlighted by the user that they would prefer a system of selecting points from a list in order to change them. An initial prototype of this screen was created in Access and it was highlighted at an early stage that the user required a system with more functionality to that of Access. At this point the tool of implementation was changed from Microsoft Access to Microsoft Visual Basic.net as discussed in Chapter 2.

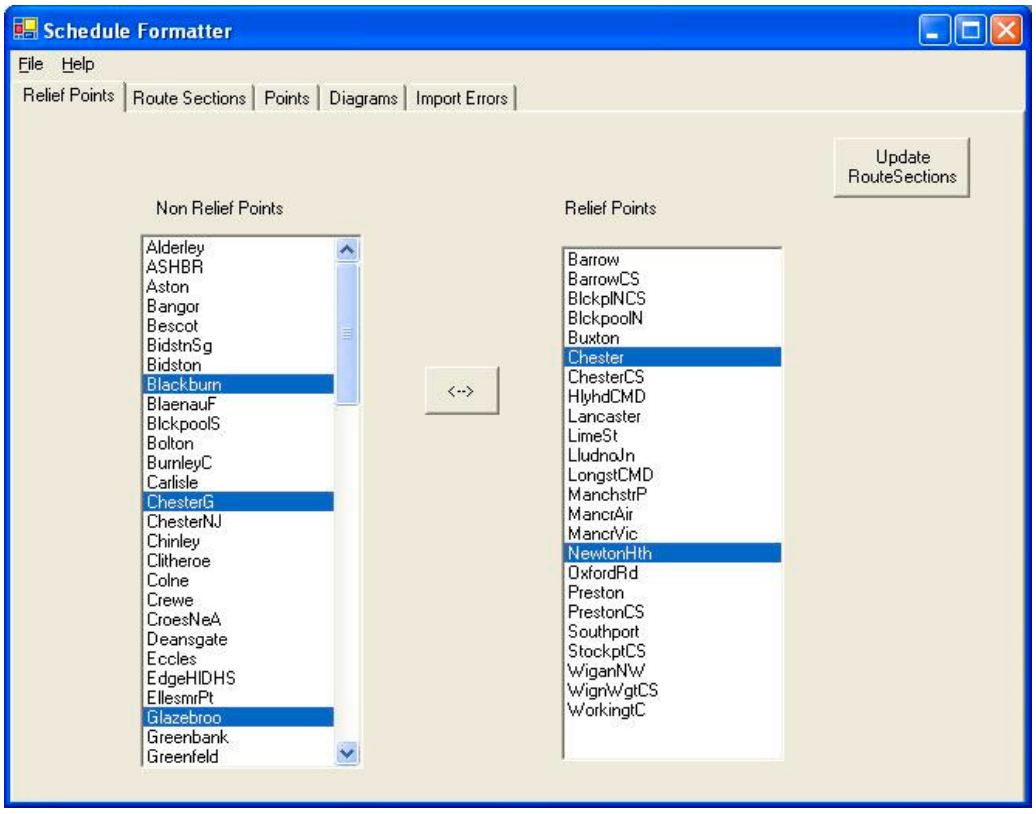


Figure 4.2 - Relief point selection screen

Two lists are used as shown in Figure 4.2, one for relief points and one for non-relief points. Any number of points can be selected from either list at any one time. On selecting the button in between the two lists the selected points are moved to the opposite list. This gives an easy to use system of selecting and de-selecting the relief points in line with the guidelines set out in Chapter 3. As the route sections will need to be updated at this point in the process a button is provided to carry out this function.

4.2.2 Route Section Knowledge Update

The route section screen shows the data created from the route section creation process. An example of which is shown below.

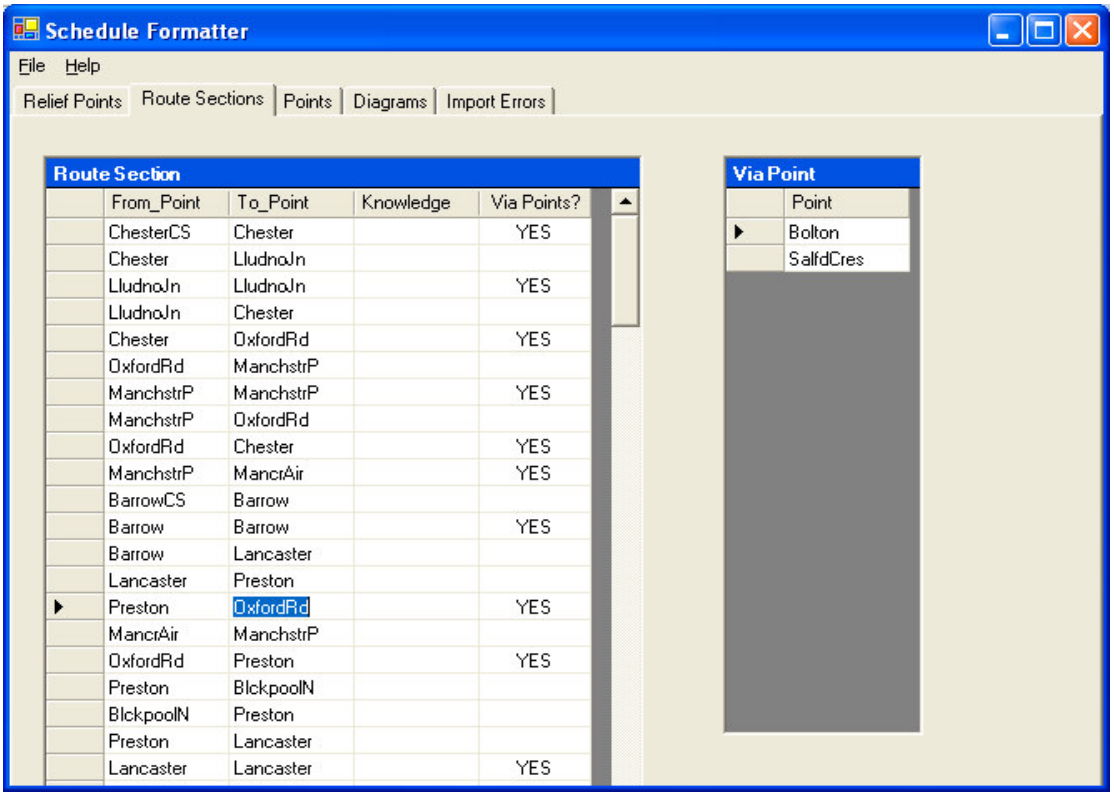


Figure 4.3 - Route section screen

The screen shows the list of route sections in one table and their related via points in another. As it cannot be seen by looking at the route section table which records are direct and which have via points, the user requested that an extra field be added. This field states whether there are via points present for the route section.

As specified in the constraints section of Chapter 2, the route sections are derived from the relief points and diagram data. The data should therefore not be able to be amended directly. All columns are therefore set to read only apart from the route knowledge, which is entered once the sections have been created.

It was requested that the user could have the option of viewing purely the records still needing route knowledge to be added. It was decided that clicking on the route knowledge column name could carry this out. This sorts the data, showing those records without the knowledge data at the top of the list.

4.2.3 Train Data Manipulation

The diagram screen represents the list of diagrams and the train details for each with any coupled trains also shown. An example of which is shown below.

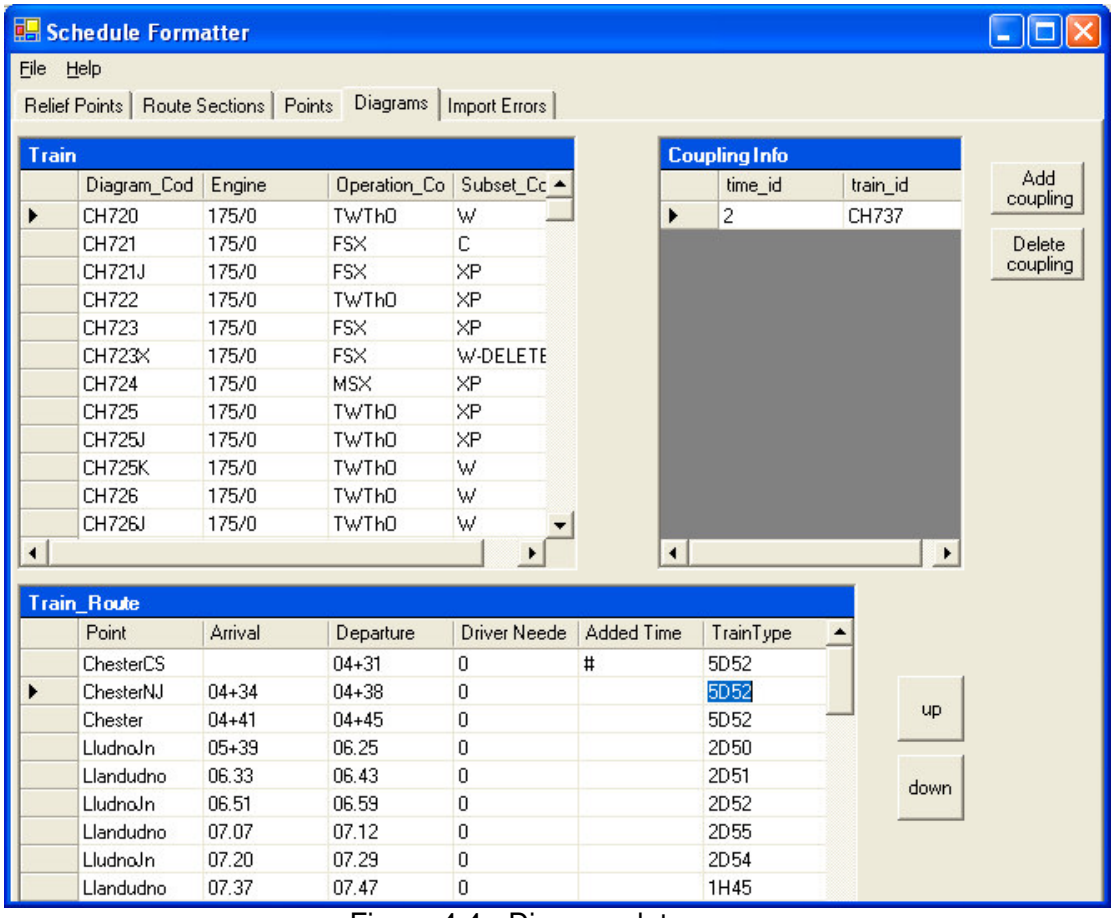


Figure 4.4 - Diagram data screen

The up and down buttons allow a journey section to be moved up or down. This means that when a new section is manually added it can be inserted at the required place without having to move all the other records manually.

As highlighted in Chapter 3 the coupling information has to have a number of restrictions so that it does not create anomalies. These restrictions are implemented as follows:

- The add coupling button produces a form with a drop down list of diagrams to select from. This allows the user to only add a current diagram reducing the possibility for errors as mentioned in Nielsen's rules in Chapter 3. When a coupling is added the related diagram is checked to see if there is a matching journey. If there isn't then an error message is raised and the record is not added. If a related journey is found then a related coupling is added by the system.
- If a coupling is deleted then the system automatically deletes the related coupling. This ensures that there is always a pair of coupled trains and no anomaly is created from having a train coupled to another but not vice versa.
- The final check is that the journey data of a coupled train cannot be updated. It must be uncoupled first before changes are made. This ensures that the journey times of the coupled trains remain the same.

4.2.4 Route Sections Function

The route sections are created by selecting the button on the route sections screen. The system looks at which points are relief points (already selected by the user from the relief point screen). It then looks at the diagram information and iterates through the list of journey times. When it finds a relief point it then looks for the next relief point. It then checks whether a route section has already been stored between these two points. If it hasn't then the system adds the route section along with the points it goes via. If the route section already exists then the system checks to see whether the via points for the two route sections are the same. If they match then the system marks the stored route section as existing and does not add the duplicate. If the two route section's via points do not match then the route section is added along with the via points.

Once the system has gone through all the diagrams it then removes all the old route sections that are no longer being used and not marked as existing. In this way the old route sections that appear again in the new information are not deleted and this results in the route knowledge stored against them not being lost as stated in the user requirements. As all the historic route sections are removed the user is asked to confirm before the process is carried out.

4.2.5 Import data

The user can choose to import data via the file option of the drop down menu. As the process will delete all current data in the system a message box is given to the user as shown below.

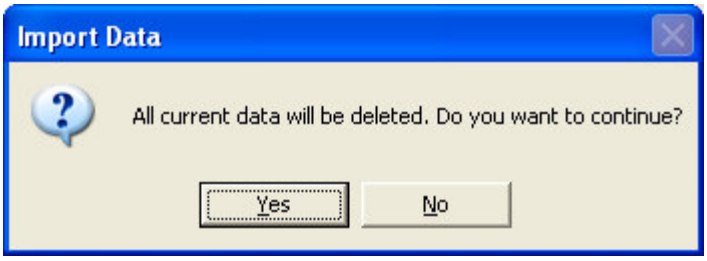


Figure 4.5 Import confirmation screen

On selecting "yes" the user is then given the option to select the file they wish to import. Visual Studio.net provides a standard dialogue box used with most Microsoft applications for selecting files. This gives the user a sense of familiarity. An example is shown below.

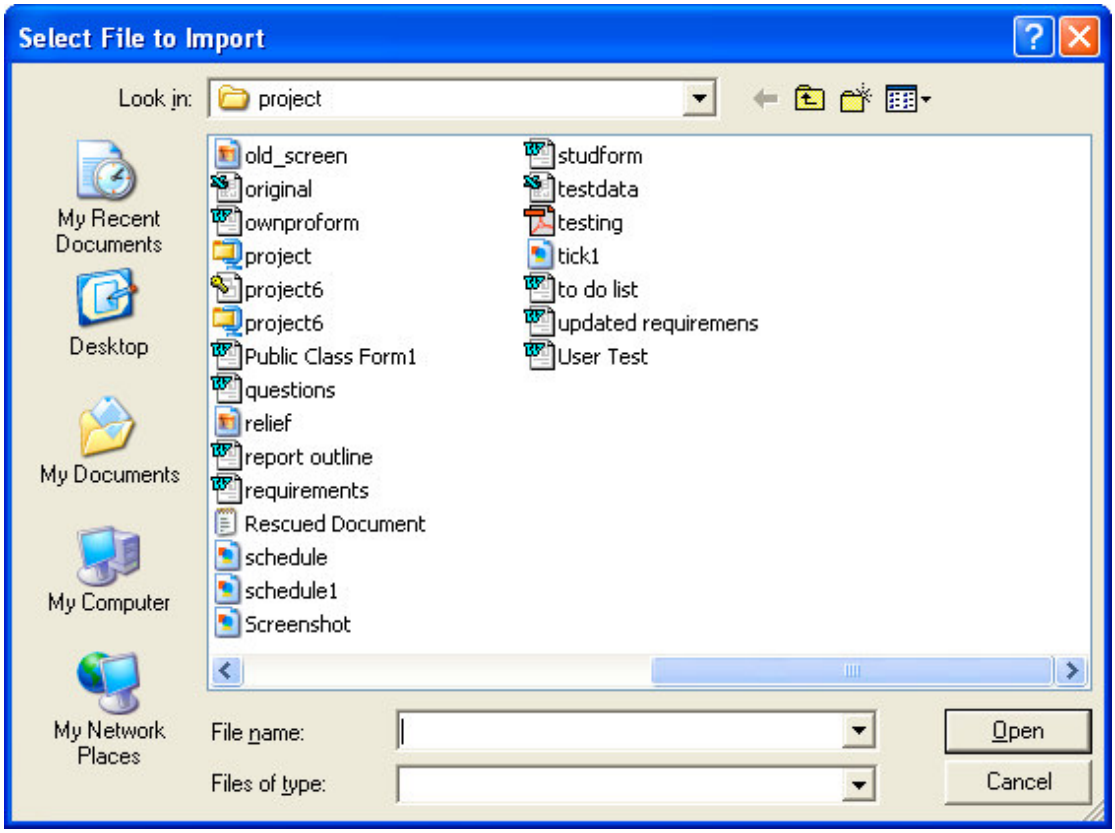


Figure 4.6 Import file selection process

The file selected must be a single sheeted excel spreadsheet. The system then imports

this sheet into the Access database as a table for reference and then loads the data into a dataset. The data is then methodically processed in the following way.
The points are added to the points table. Those points where a diagram starts and ends are automatically selected as relief points. Each diagram in the spreadsheet has a heading giving the diagram information. This row is used to find the start and end of a diagram. The system checks to see whether the diagram already exists. If it does an error message is shown to the user as follows. The user is then given the option to view any further errors or to ignore them.

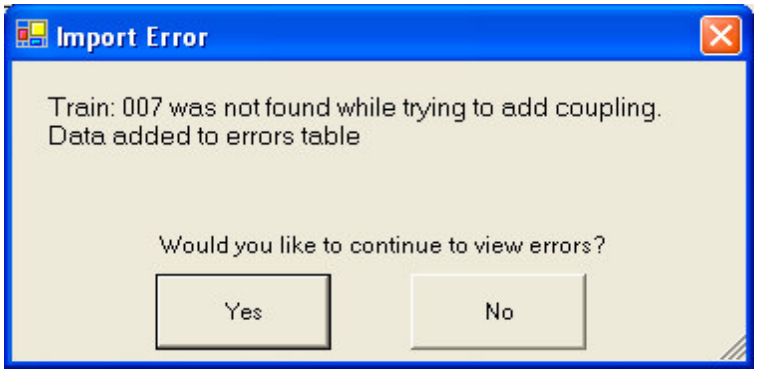


Figure 4.7 - An example of an import error

Any errors resulting from the import are shown on the import error screen shown below.

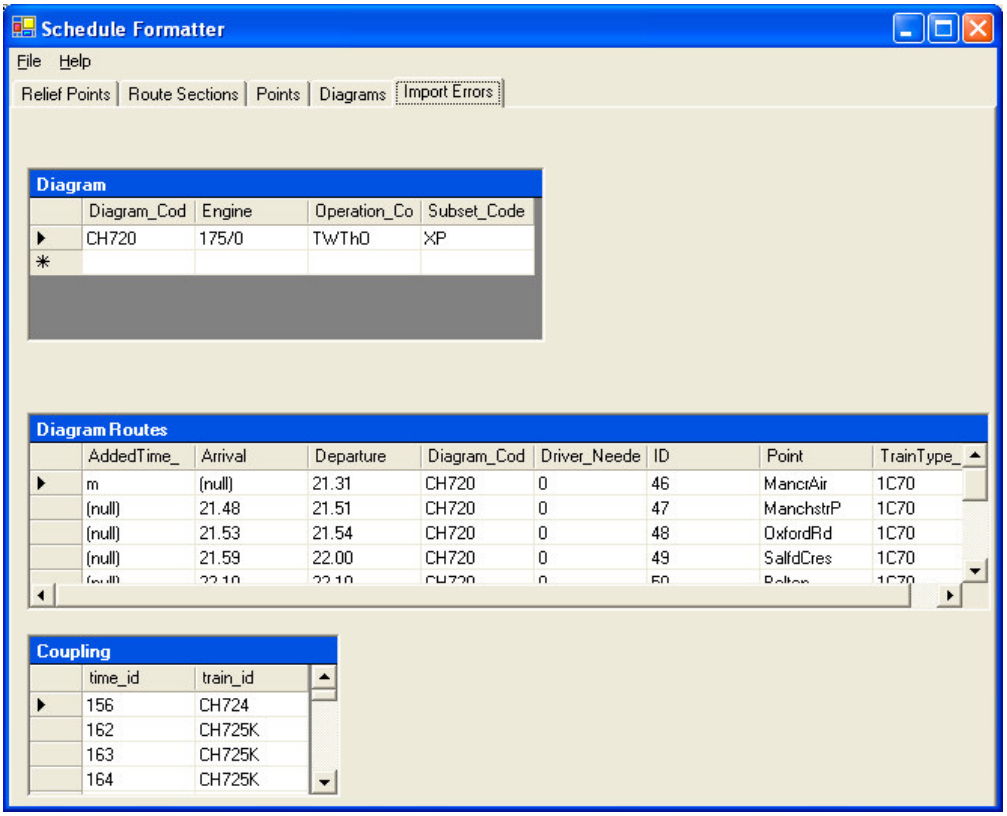


Figure 4.8 - The Import Errors information screen

This allows the user to see which records were not added to the database and can then allow the user to decide what to do with this information.

Once the diagrams and their routes have been added the coupling information is then added. This results in the raw data having to be iterated through twice, which does slow the process down. This was done in order to check that the related train data matched up with the coupled train. This could only be done once the diagram data had been added to the system.

Once the import has take place a message is given to the user stating that the import has been completed.

4.2.6 Export Data

Once the data has been imported, relief points chosen, route sections created and route knowledge added the data is ready to be exported. Selecting export from the drop down menu carries out this process. If the required processes have not been carried out when the export function is selected the system will inform the user as shown below. This stops any errors taking place and informs the user of the correct action required.

Figure 4.9 - Route section error message

If all the required processes have been carried out then the system will ask the user to select a file to export to. The system uses a similar screen to that used when selecting a file to import, this creates consistency in the system as required in the design of the interface stated in Chapter 3.
The file is then exported. The schedule system requires that the exported data be set out in an exact specified format. The information as to how the data should be set out was obtained both from the user and from existing example outputted files.

Once the export has completed the system informs the user and asks if they would like to view the exported data. If they do then the file is opened in notepad as shown below.

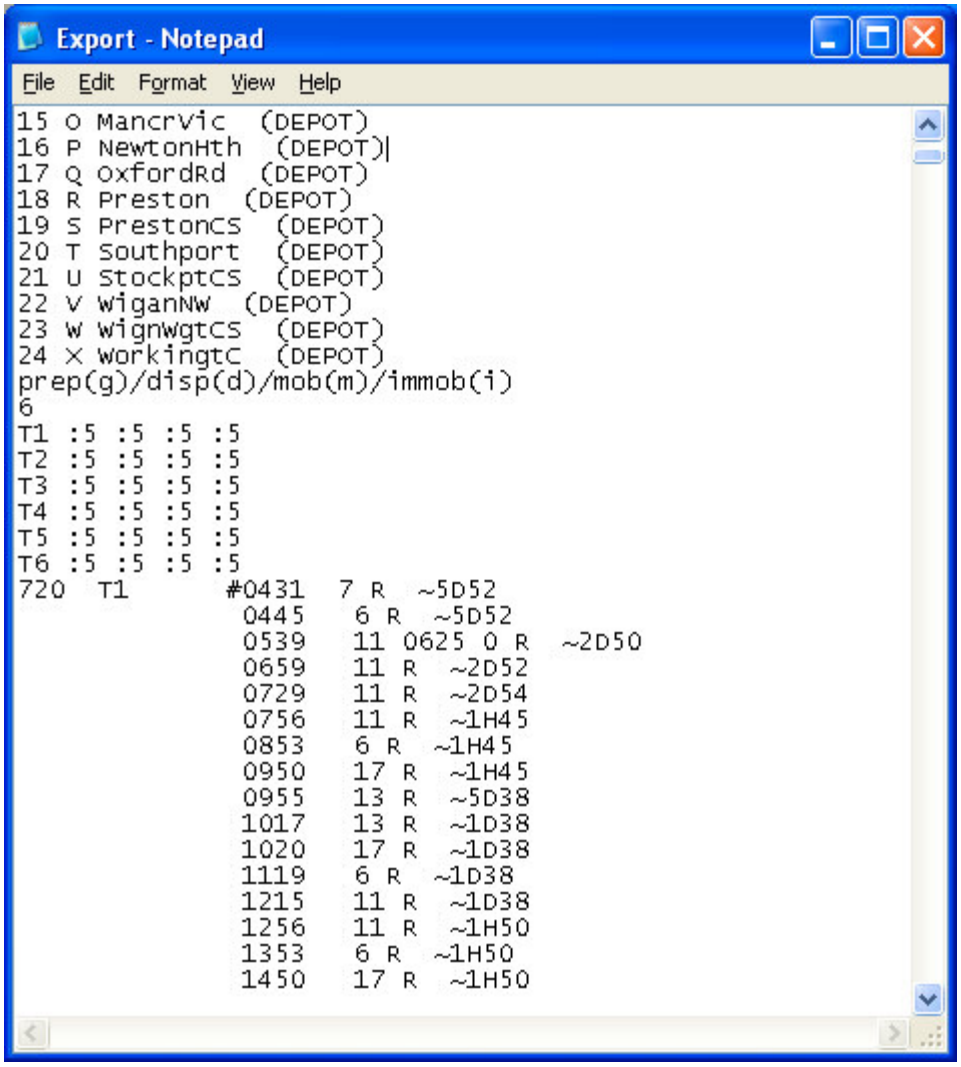


Figure 4.10 - An example export data file

4.2.7 Process Form

The process screen shown in Figure 4.11 below was a late addition to the system and requested by the user so that they can assess what stage in the processing of data they are at as defined by Figure 2.1 in Chapter 2. The screen uses the familiar visual aid of the green tick and red cross to highlight the processes completed and those needing to be carried out.

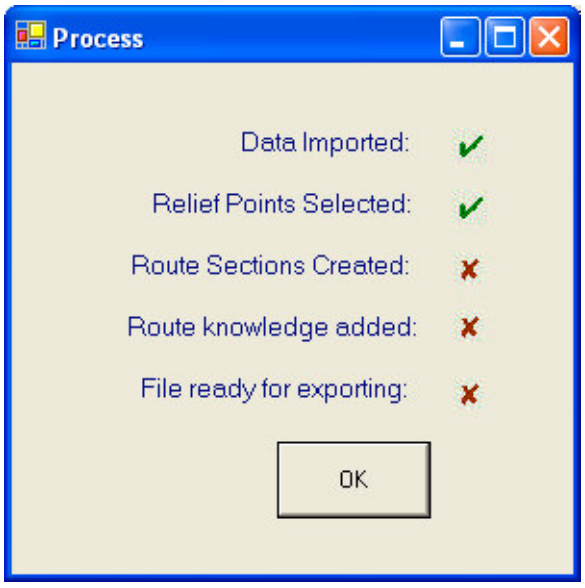


Figure 4.11 - The process screen

4.3 User Manual

The user manual gives the users of a system a resource to be able to refer to in order to gain an understanding as to how to use a system. Nielsen (1993) highlights the fact that "one important aspect of help and documentation, whether online or a hardcopy is the quality of the writing especially including the structuring of the information and readability of the text." The structuring of schedule formatting software documentation will therefore be around the processes and functions built within the system this is also helpful as Schneiderman (1998) states that a manual should be tailored to the "accomplishment of specific goals." The documentation is therefore based around completing each of the processes defined in the activity diagram of Chapter 2.

A copy of the manual can be found in Appendix B.

4.4 Installation

One of the useful features of Visual Studio.net is its allowance for deploying a project. Many options are allowed such as whether a shortcut is placed on the user's desktop or if it should appear in the users start menu. On deploying the project a setup file is created. This is very simple to put on a CD and can then be easily distributed. The setup program provides a simple process for the user to install the application on their computer. Figure 4.12 below shows an example of this.

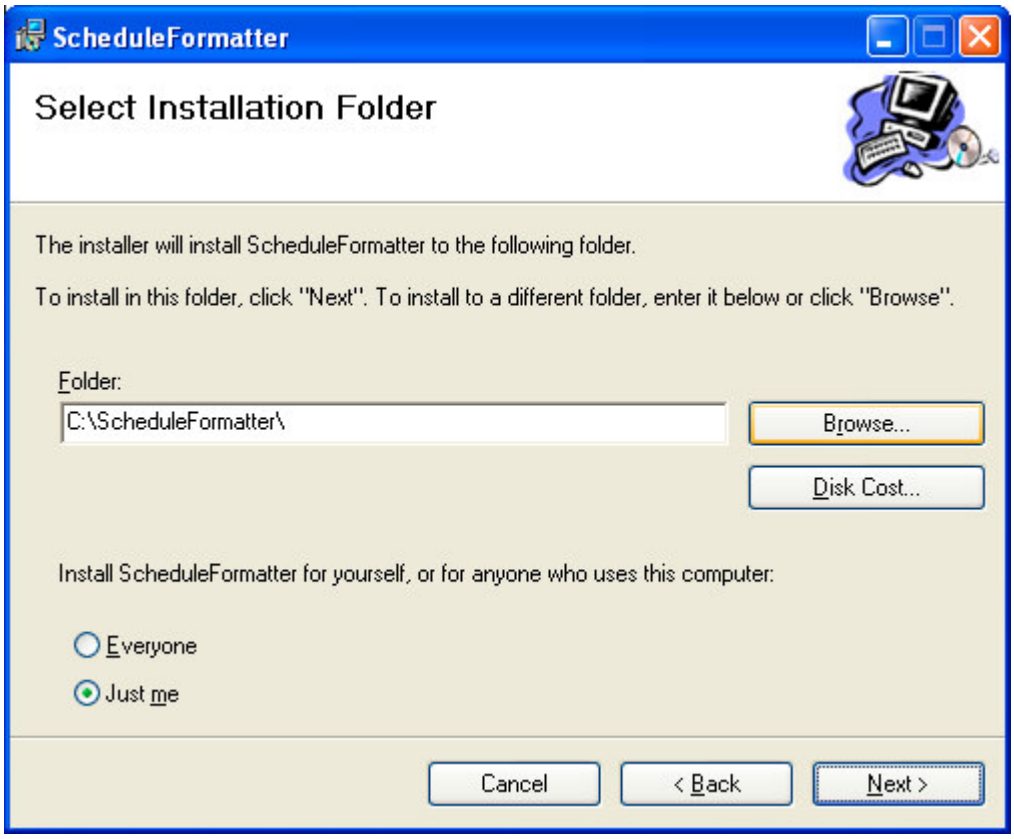


Figure 4.12 - The installation screen

The link between the Visual Basic.net interface and the Access database is dynamic. This therefore means that the user does not need to carry out any additional setup in defining where the database is held.

5. Testing

This chapter looks at the different ways in which a system can and should be tested. In particular it examines how the schedule formatting software was tested and the changes made as a result.

5.1 The Requirements of Testing

Rees et al (2001) highlight the need for testing in order to reduce the level of risk to a producer and users business and to increase confidence that a product will provide an acceptable level of risk. They state some of the main areas to consider in the field of testing as follows:

- How to quantify reliability.
- How to design tests.
- Which tests should be performed given time, cost and resource constraints.
- What tests should be rerun following corrections to rectify faults.

The software product evaluation standard ISO 9126 from the International Organisation for Standardisation (2001) states that reliability can be measured by the three following criteria:

- **Maturity**
  The capability of the software product to avoid failure as a result of faults in the software. This is covered more fully in the following two points
- **Recoverability**
  This can be described as how a system recovers from a crash. Issues involved are the possibility of corrupted files and loss of data. It could be said that a fully tested system should not have to deal with the issue of recoverability however some situations are outside the control of the system such as hardware failure.
- **Fault Tolerance**
  This can be described as the ability of a system to degrade gracefully when unexpected software or hardware errors occur. This has been discussed in the user interface section of Chapter 3. However, the system needs to be tested that the code used to catch errors was implemented correctly and that no errors were missed.

The testing of the system should therefore be most concerned with highlighting areas where faults occur. Fixing faults where possible or adding code to allow the system to degrade gracefully if errors are outside of the systems control. The further questions of consideration given by Reese will be looked at in the next section.

5.2 Test Design

Figure 2.2 in Chapter 2 highlights that testing should be carried out as part of the process of creating each operation product. In light of this basic testing was carried out at each of these stages with a more in-depth testing of the whole system planned once all the operational products had been produced. Mosley and Posey (2002) state that designing the tests is the most important part of the testing process and highlight the need for the tests to test the requirements of a project.

Jacobsen et al (1999) define two types of test that can be carried out on a project:

- **Black Box Test**
  A black box test, tests the externally observable behaviour of a system. It is therefore concerned with the users interaction such as the input and expected output of the system.

- **White Box Test**
  A white box test, tests the internal interactions between components of the system. It therefore tests what the user cannot see and looks more at the interaction of the processes within the system.

Jacobson et al (1999) highlight that a test case should be evolved around the use cases and state that "to be complete, the test case must specify the input, expected result and other conditions relevant for verifying the scenario of the use case".

This approach was therefore employed when testing the schedule formatting software. The user was given a set of tasks based on the use cases specified in Chapter 2. The headings for the test were as follows.

- **Test Name**
  The name of the procedure to be carried out relating directly to the use cases.
- **Procedure**
  The procedure involved in completing the use case. This was added to ensure that the user was carrying out the use case in the correct manner and if not what the errors were as a result of.
- **Result**
  The result produced from the procedure.
- **Expected Result**
  The result expected from the system.

For white box testing Bennett et al (2001) state that "Test designers will examine the sequence diagrams to find combinations of user input, output and system start state that will test the interaction between the participating objects." Tests were therefore created from the activity diagram shown in Figure 2.1 in Chapter 2. This involved repeating processes such as the import function and carrying out a stage such as the creating of route sections then amending the relief points and recreating the route sections.

The most important part of the testing was that of the export facility. The main purpose of the software is to provide the exported data for the scheduling software. The final test was therefore to check whether the system produced the same data as the old system and as to whether the data would be accepted by the scheduling software.

5.3 Test Implementation

The actual testing was carried out by one of the end users Ignacio Laplagne, a PhD student in the School of Computing who tested the black box areas of the system. When the user came to testing, they preferred not to use the given system and found it easier to write a list of their findings. Although this was not as simple to analyse as the given form the information that would have been shown on the form was still retrievable from the users notes.

The designer carried out the white box testing. This was a very useful process and highlighted a number of problems which would probably not have been found

otherwise. This included issues such as storing the data twice in the dataset when importing the data two times in a row. The user would probably not carry out this action very often but by applying the test was uncovered.

Jacobsen et al (1999) highlight another test known as the installation test. This is only relevant to customised software rather than mass market and tests that the software works correctly on the users system. This test highlighted a number of small problems that were duly rectified.

The final test as mentioned above was that of the export facility. A number of amendments were needed in this area as the schedule system is very specific in how the data should be represented. Issues such as blank lines and spacing of data had to be corrected before the data was accepted.

6. Evaluation

This is the final chapter and gives a summary to the project and evaluates the final result of the project and as to whether it meets it's initial requirements.

6.1 Evaluation of the system

The original requirements for the project were to create a system for the manipulation of train data. Incorporating a database for holding the information, an interface for each of the processes involved in the data manipulation process, a set of functions for manipulating the data and exporting it and finally creating a user manual to be used along side the system.

These requirements have all been met. The database was designed and normalised as shown in the implementation chapter. The final database is not fully normalised but reasons were given for this and constraints were included to ensure referential integrity of the data.

The system incorporates a tool for the user to import the data that is simple to use. The system shows the user clearly any records that were not imported. The system does not allow these records to be imported at a later date, which would have been a nice addition but was not possible due to time constraints. The system also has an easy to use system for selecting relief points and a data extraction function for creating route sections based on the selected relief points.

The system provides a function for allowing the user to export the data. Allowing the user to select the file they want to export to. The data is automatically exported in the format required by the scheduling software.

The system allows the user to manipulate the train data with a number of extra functions added such as changing the order in which trains stop.

Finally the system includes a manual showing clearly how to accomplish each of the tasks in formatting the data.

6.2 Possible Improvements

There were a number of possible improvements that were highlighted through the progression of the project these have been listed as follows:

- **Import errors**

  As mentioned in the previous section the software highlights those records that contain errors and are not imported. These are then stored in a separate table that the user can view. It would have been useful for the user to then be able to amend these errors and be then given the option of adding them to the imported data. As this is not included the user would have to amend the data at the source which creates extra work.

- **Route knowledge source data**

  A late request for the system was to include the data used by the user when assigning the route knowledge to a route section. This data is not linked to any of the data currently used within the system but would be useful to the user to have to hand when carrying out this process. Sadly due to time constraints this function was not added to the system.

- **Data manipulation restrictions**

  The scheduling software will not accept data with any inconsistencies such as when the time of a train arriving at a station is before it departed from the previous station. Another inconsistency might be for example if the train has an additional time given to it for starting up the train when it is already started. It would be helpful for the system to highlight these inconsistencies in the formatting software so that the user would be able to see any problems before the data was applied to the schedule software.

6.3 Conclusion

In conclusion, the methodology was ideal for the project and allowed the work to be segmented into smaller units of work, which also allowed or user feedback. The chosen technology gave a good end product but took longer to produce and reduced the amount of features produced. Being able to run the data through the scheduling software ensured that the outputted data was of a correct format. The time taken to test and make corrections took longer than was scheduled and more time should have been

allocated for this at the end, rather than relying on minimal testing throughout the project.

Overall however, the project has been a success, there are a number of areas where it could be improved but the core requirements have been met with a number of features added such as the storing of import errors.

Bibliography

Behforooz, A, Hudson, F, (1996) **Software Engineering Fundamentals,** Oxford University Press

Bennett, S, Skelton, J, Lunn, K, (2001), **Schaum's Outline of UML**, McGraw Hill

Bilgin, P, (2002), **Visual Basic.Net Database Programming,** Sybex

Carroll, J, (2001), **Human-Computer Interaction in the New Millennium,** Addison-Wesley

Elmasri, R, Navathe, S, (2000), **Fundamentals of Database Systems,** Addison Wesley Longman

European Commission, (2001), **CORDIS: TAP for Transport: Karen,** URL: http://www.cordis.lu/telematics/tap_transport/research/projects/karen.html, [24 April 2004]

ISO/IEC. (2001), **International Standard ISO/IEC 9126-1. Software engineering -- Product quality -- Part 1: Quality model**. International Organization for Standardization / International Electrotechnical Commission. Geneva.

Jacobson, I, Booch, G, Rumbaugh, J, (1999), **The Unified Software Development Process,** Addision Wesley

Johnson, O, (2002), **IN21: Object Oriented Analysis and Design,** University of Leeds

Kwan, R., (2004), **Handbook of scheduling,** CRC Press

Lauden, K, Lauden J, (2002), **Management Information Systems Managing the Digital Firm,** Prentice Hall

Mosley, D, Posey, B, (2002), **Just enough software automation,** Prentice Hall

Nielsen, J, (1993), **Usability Engineering,** Academic Press

Pressman, R, (2000) **Software Engineering A Practitioners Approach 5th Edition,** McGraw-Hill

Redmill, F, (1997), **Software Projects: Evolutionary vs. Big Bang Delivery,** Wiley, p12-17

Reese, G, (2003), **Java Database Best Practices,** O'Reilly

Shackel, B, (1997), **Human Computer Interaction - Whence and Whither?,** Journal of the American Society for Information Science

Schneiderman, B (1998), **Designing the User Interface 3rd edition**, Addison Wesley

Appendix A: Reflections on the Project

This appendix gives a reflection on the project and the lessons that can be learnt for future projects.

On the whole I feel the project gave me a good understanding of the issues faced when carrying out a project from start to finish and gave me a practical understanding of the issues learnt throughout my degree such as analysis, methodology, database design and programming.

The first lesson I learned was the need for a full understanding of the requirements of a project. I had carried out a number of interviews with the user to obtain an understanding of what was required but had also incorrectly assumed a number of factors such as the meaning of a route section. This might have been avoided if I had gained feedback on the requirements I had obtained from the user who might have been able to highlight any discrepancies in understanding.

The methodology I chose for the project was a good choice and highlighted the need for a structured methodology throughout a project. This enabled me to structure the work and gave me guidelines to work to which was vital for the completion of the project. It showed that for any project to be successful it needs some kind of methodology to be chosen at the outstart, it is also important for the designer to keep to the methodology throughout the project or else there is not much point choosing one in the first place.

I found that the choice of technology can need more forethought than first realised. Although I investigated the different technologies available I did not fully appreciate the time taken to learn a new tool such as Visual Basic.net. Although the user required more functionality I feel I could have provided more features had I continued using Microsoft Access. When choosing a technology, consideration should therefore be given to the cost in learning a new technology. When only having a fixed amount of time for a project, the time taken to learn a new language can mean that less time is available for the development of some of the required features.

Appendix B

Schedule Formatter

User Manual

Contents

Installation Procedure

Schedule Formatter comes with its own installation package. It is therefore very easy to install by following the points below.

- Select the setup file

- The following screen should appear

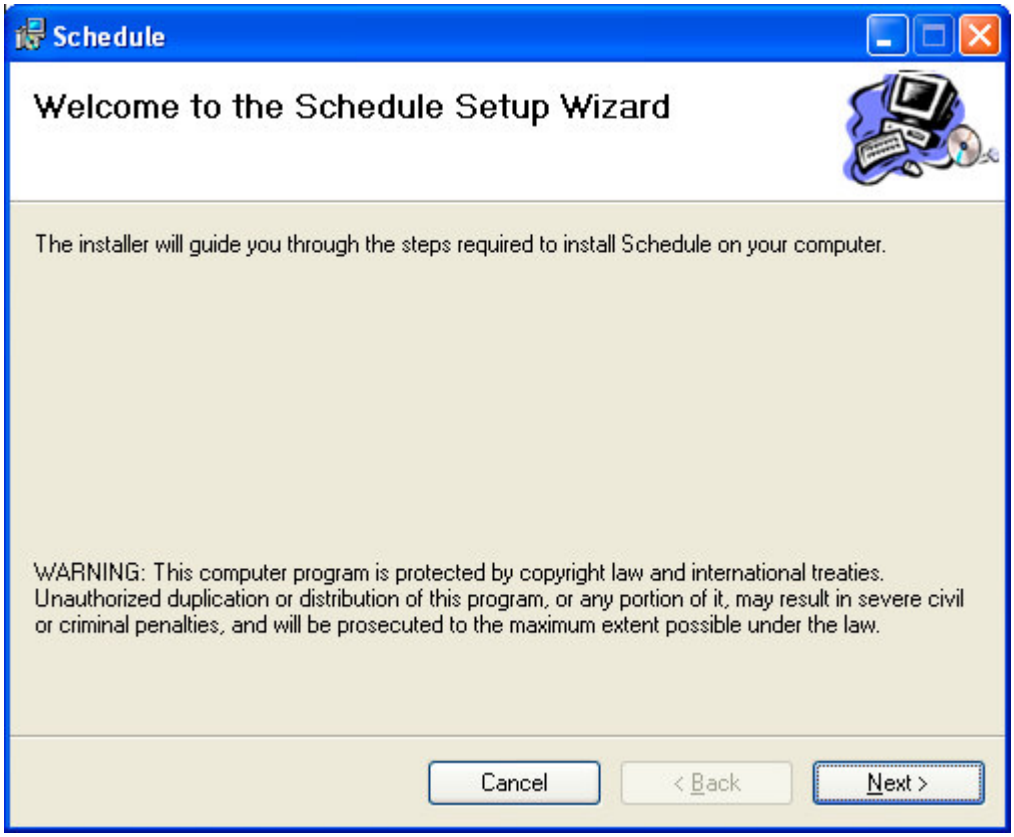


Figure 1 - Installation Screen

- Click 'next'. On the next form you will be asked where you would like to store the Schedule Formatter software. The default path is C:\ScheduleFormatter. If you would like to install the software to another directory this is easily achieved by selecting browse and selecting the desired path. Once selected click next.

- The final screen asks for confirmation that you wish to go ahead and install the software. If you are happy to go ahead then select next and the software will be installed in your specified directory.

Welcome to Schedule Formatter

Welcome to Schedule Formatter. The software is designed to let you format your schedule data in order to allow you to submit your data for train driver scheduling.
To achieve this goal there are a number of processes that have to be gone through in order to be able to export your data. A simple form shows you the stages as shown below:

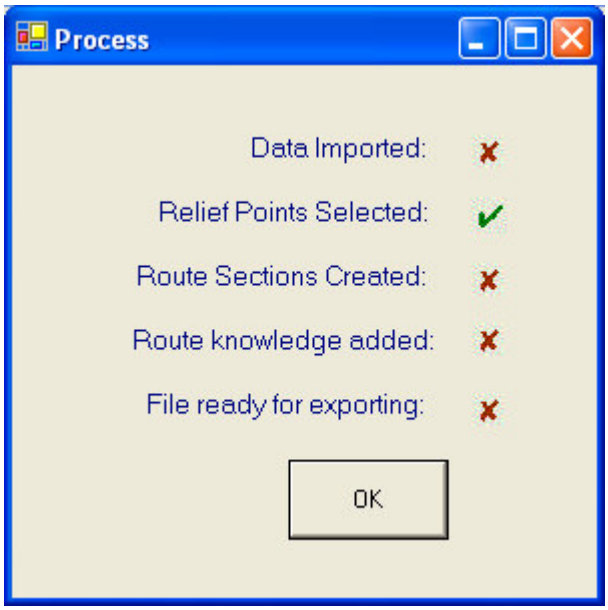


Figure 2 - Process Screen

At any point whilst using the software you can select the form to see which stage you are at. The green ticks highlight the processes that have been completed. The red crosses show the processes still to be completed.

The process form can be selected from the menu as shown below

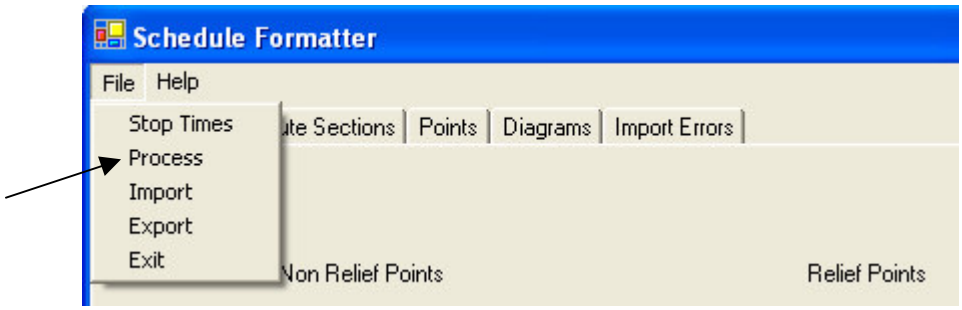


Figure 3 - Process Selection

The rest of this manual will go through each of these sections in turn showing how the system can be used to complete each process.

Importing Data

When first installing Schedule Formatter there will be no data available in the system. Data can be imported into the system at any point. This section looks at how to achieve this.

The import command can be found on the file menu as shown below:

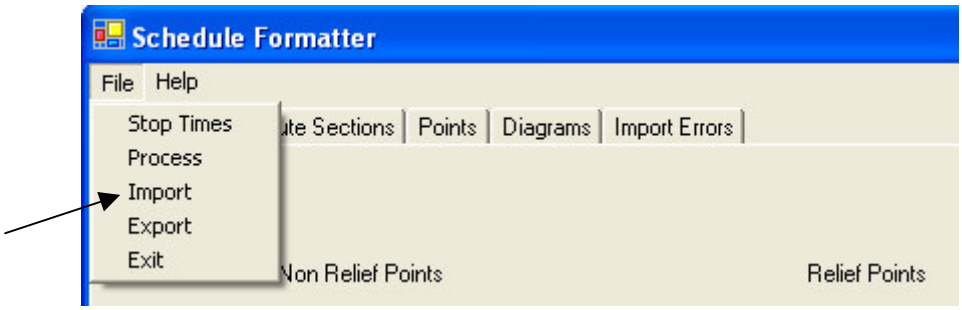


Figure 4 - Import Selection

Once selected the system will give you a warning stating that all data within the system will be deleted upon importing new data. It will give you the option to proceed or to cancel the import.

If you have selected 'proceed' you will then be asked to select a file of data to import. The file you select must be an excel spreadsheet.

If there are any errors in the data then the system will highlight these as shown below:

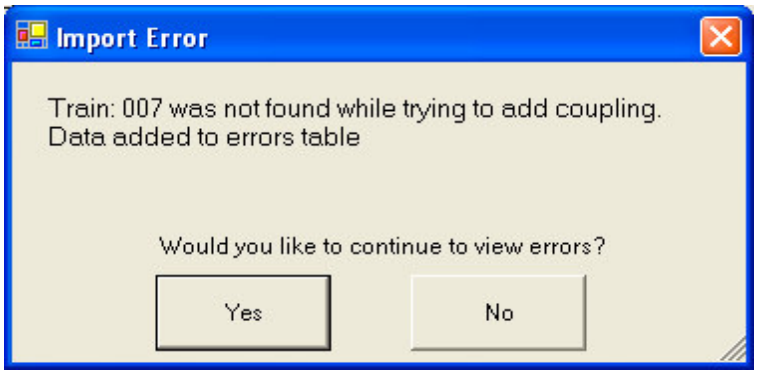


Figure 5 - Import Error

Any errors shown will not be added to the database but will be shown in the errors section (refer to the import errors section for more details).

If the data is imported successfully then a message will be shown confirming the importing of data.

If there is a problem with the data such that it cannot be read at all then an error message will be shown.

Reasons for this could be as follows:

- You have selected a non excel file type
- Your import file is already open by another process
- Your import file is not in the correct format

Selecting Relief Points

Once data has been imported to the system you will see that the relief points fill up as seen in the picture below. Those points where a diagram starts or ends are automatically selected as relief points.

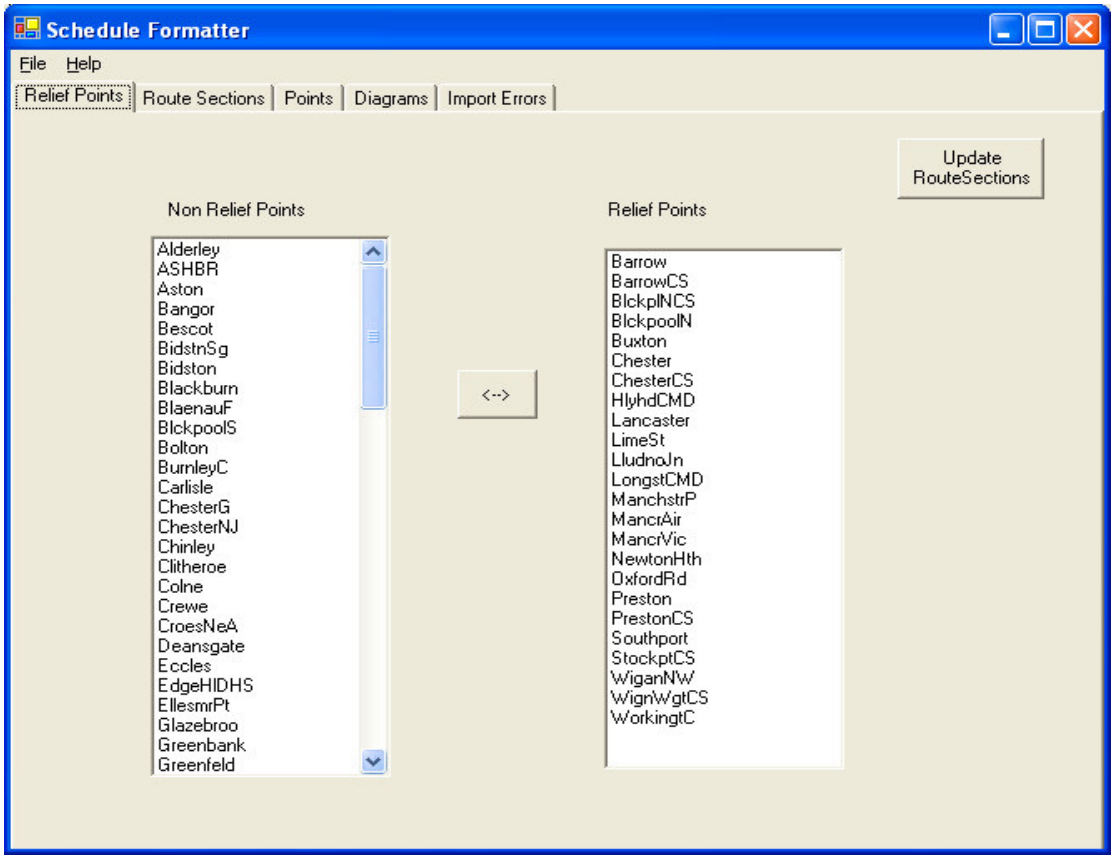


Figure 6 - Relief Point Selection Process

To move points from one list to the other you can select as many points as needed from either side by clicking on the point. To move the point to the other list click the double ended arrow in between the two lists.
When the button has been selected you will see that the selected points have moved to the opposite list.

Once you have selected your relief points you are now ready to create the route sections. This process can be carried out at any stage, but it should be noted that any new route sections created will need route knowledge adding.

The button titled "update route sections" in the top right of the "Relief Points" screen as shown in Figure 6 is used to create the route sections. A message box will confirm the route sections have been created once the process is complete.

Adding Route Knowledge

The route knowledge can be added on the Route Section screen, accessed by selecting the "Route Sections" tab as shown by the arrow on Figure 7 below.

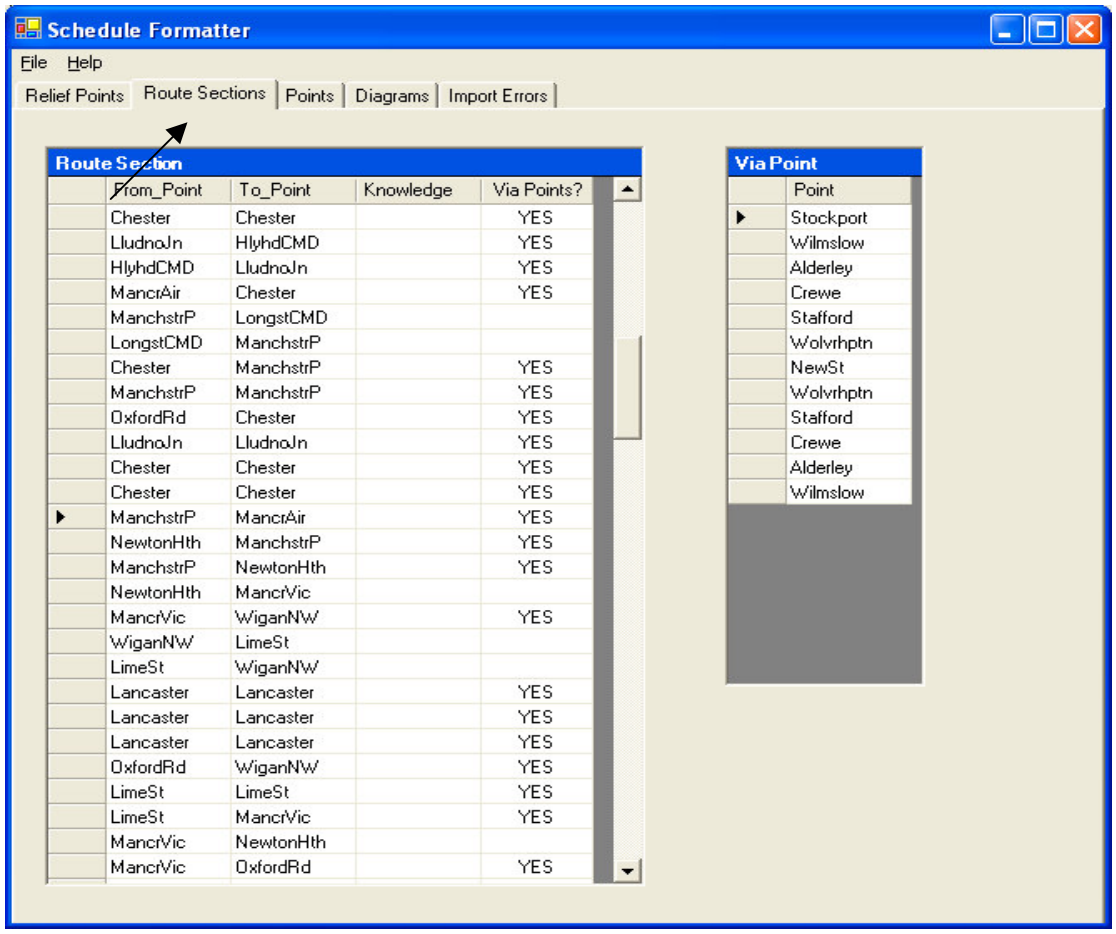


Figure 7 - Route Section Screen

The main table shows the route sections with the third column used for updating the route knowledge. The via points for each route section are displayed on the right hand side table.

Stoppage Times

The time taken for each of the set types of stoppages can be selected from the file menu under 'Stop Times'. This will open a form as shown below.

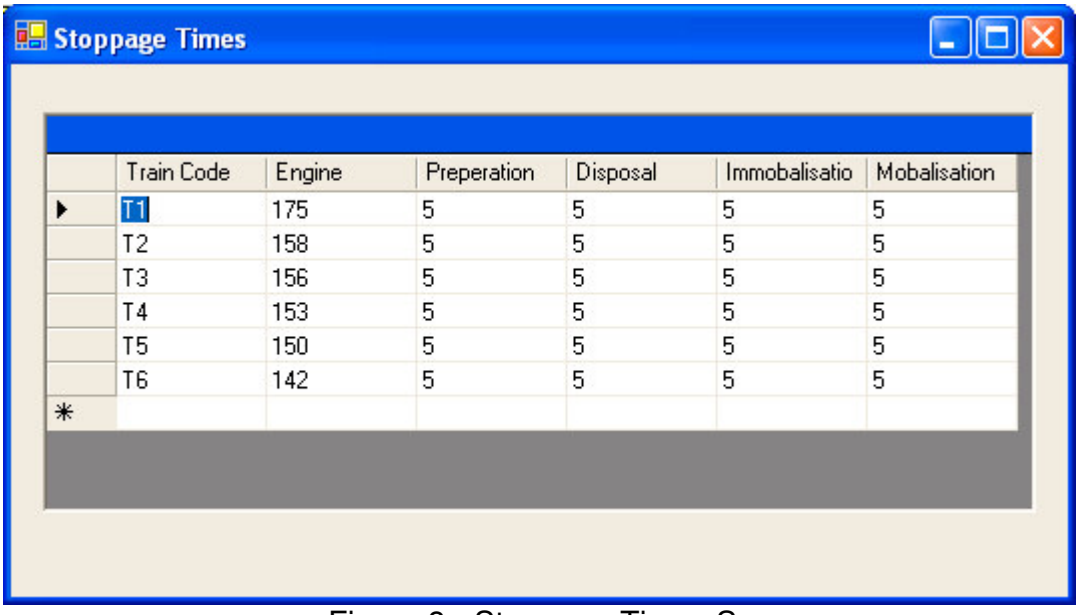


Figure 8 - Stoppage Times Screen

As you can see from Figure 8 a table is opened where the times for each engine can be updated. This is not part of the data processing and can therefore be updated at any point. The data is not affected by carrying out the import function.

Exporting the Data

Once all processes have been completed the data will be ready to be exported. This will be represented on the process form by the fact that there will be a tick next to "data ready for export".

The export function can be accessed from the file menu as shown below:

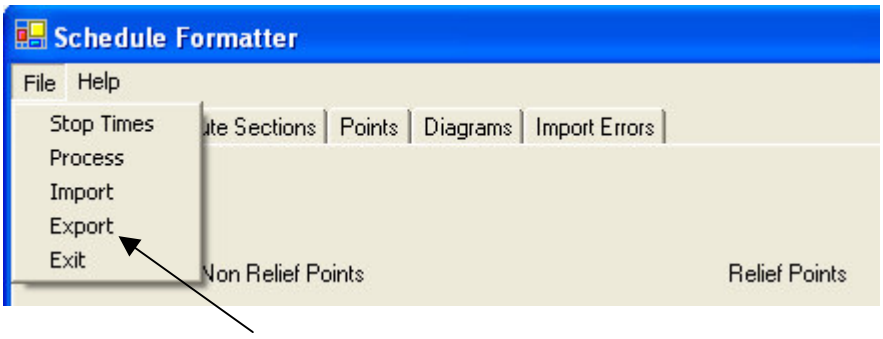


Figure 9 - Export Selection

Upon selecting the export option the system will ask you where you would like to export your file to. Select a file or enter a new name and select OK.

The system will then ask you to give an optional title to your exported file as shown below

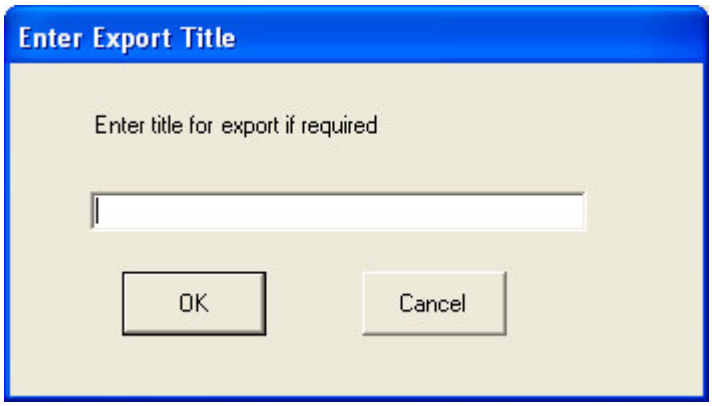


Figure 10 - Giving the Export a Title

The title you enter will appear at the top of the exported file. If you would like to omit this then select cancel. The export process will then continue without using a title.

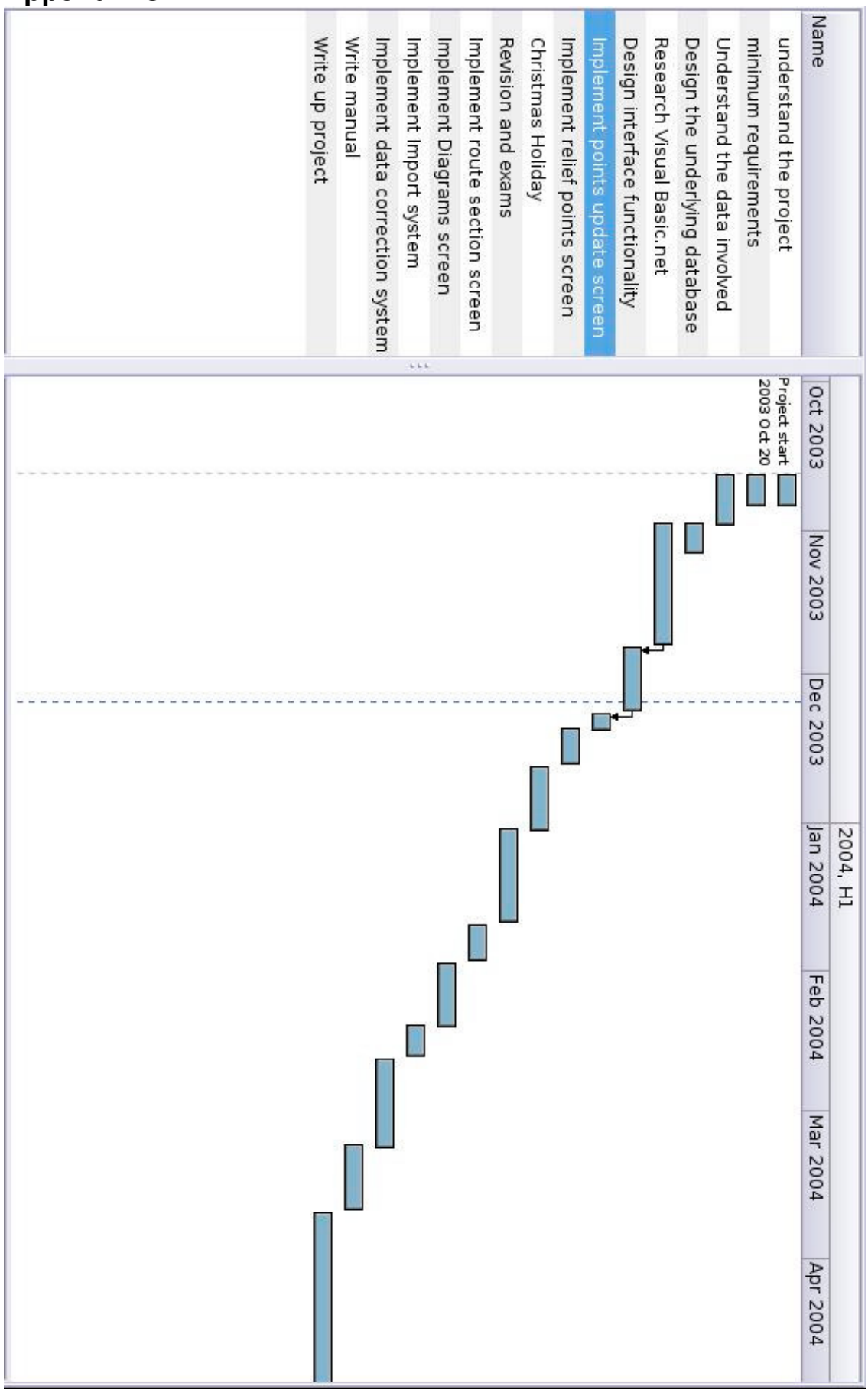
If the export is successful a message box will appear asking if you would like to view the exported file. Click OK to view the file or select cancel if this is not required.

Un-installing the Software

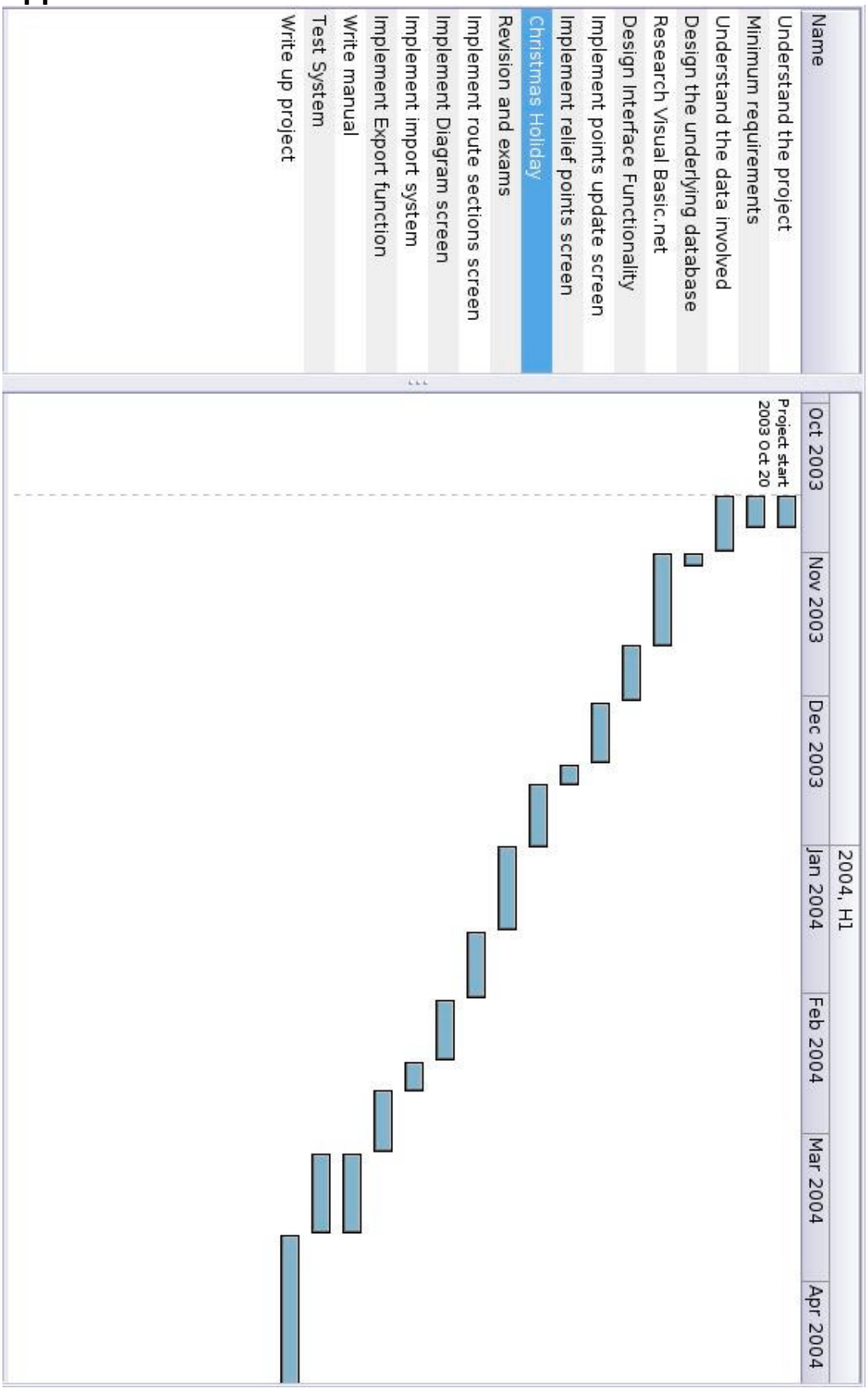
Un-installing the software can be carried out via the windows control panel. The steps to complete this process are as follows:

- Select control panel from the start menu.
- Select Add/Remove programs.
- A list of installed programs will appear. Select the Schedule Formatting software.
- Select remove making especially sure you have the right program selected.
- The computer will then remove the software and any desktop links to it.

Appendix C


Name
understand the project
minimum requirements
Understand the data involved
Design the underlying database
Research Visual Basic.net
Design interface functionality
Implement points update screen
Implement relief points screen
Christmas Holiday
Revision and exams
Implement route section screen
Implement Diagrams screen
Implement Import system
Implement data correction system
Write manual
Write up project
Project start
2003 Oct 20
Oct 2003
Nov 2003
Dec 2003
2004, H1
Jan 2004
Feb 2004
Mar 2004
Apr 2004

Appendix D


Name
Understand the project
Minimum requirements
Understand the data involved
Design the underlying database
Research Visual Basic.net
Design Interface Functionality
Implement points update screen
Implement relief points screen
Christmas Holiday
Revision and exams
Implement route sections screen
Implement Diagram screen
Implement import system
Implement Export function
Write manual
Test System
Write up project
2004, H1
Oct 2003
Nov 2003
Dec 2003
Jan 2004
Feb 2004
Mar 2004
Apr 2004
Project start
2003 Oct 20

Appendix E

ManchstrP	00.02	00+04	5J50	359 \317					
Mayfield	00+06	00+10	5J50	359 \317					
ManchstrP	00+12	00+12	5J50	359 \317					
OxfordRd	00+14	00+14	5J50	359 \317					
SalfdCres	00+20	00+23	5J50	359 \317					
MancrVic	00+30	00+30	5J50	359 \317					
NewtonHth	00+37							d	
DIAGRAM	NH326	156/0	TWThO		W				
NewtonHth		04+58	5D32	315 \326			x		
MancrVic	05+05	05+06	5D32	315 \326			x		
SalfdCres	05+12	05+16	5D32	315 \326			x		
OxfordRd	05+20	05+20	5D32	315 \326			x		
ManchstrP	05+26	05+26	5D32	315 \326			x		
Mayfield	05+28	05+37	5D32	315 \326			x		
ManchstrP	05+39	05.41	1D32	315(F)\326			x		
OxfordRd	05.42	05.42	1D32	315(F)\326			x		
NewtonLe\	06.00	06.00	1D32	315(F)\326			x		
WarrgtnBC	06.09	06.10	1D32	315(F)\326			x		
Chester	06.40	06.48	1H42					m	
WarrgtnBC	07.14	07.15	1H42						
NewtonLe\	07.24	07.24	1H42						
OxfordRd	07.45	07.47	1H42						
ManchstrP	07.49								
DIAGRAM	NH326J	156/0	TWThO	07.51	S				
ManchstrP		07.49	2D42						
Stockport	07.59	08.00	2D42						
Chester	09.13	09.57	2H85						